MAGAZINE

BSD

FOR NOVICE AND ADVANCED USERS

HANDLING KERNEL PANIC

INSIDE

REACTING TO PANIC: HOW TO CONFIGURE THE SYSTEM TO HANDLE CRASH DUMPS

MAHESHABSD SERVER: MYSQL AND WORDPRESS IN FREEBSD

FREEBSD PROGRAMMING PRIMER: HOW TO EMBED CSS AND JAVASCRIPT IN PAGES

HARDENING FREEBSD WITH TRUSTEDBSD AND MAC CONFIGURATION OF MAC_IFOFF, MAC_PORTACL, AND MAC LOMAC MODULES

VOL.7 NO.3
ISSUE 03/2013(44)
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.

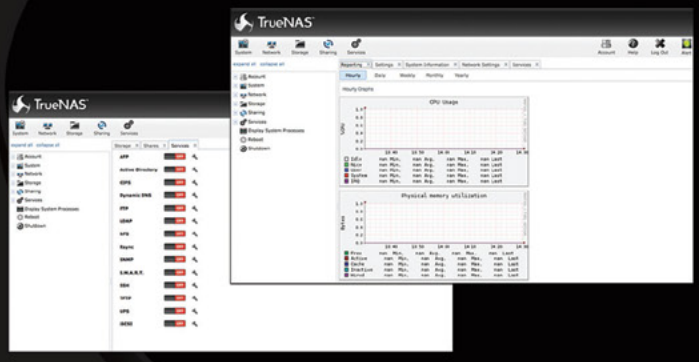
TrueNAS™ Storage Appliances: Harness The Cloud

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

	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	Six	Quad	Dual
10 Gigabit NICs		Dual*	Quad*		Dual*			Dual
Max Main Memory	48Gb	96Gb	192Gb	48Gb	192Gb	48Gb	96Gb	192Gb
Max Capacity	220TB	500TB	1.5PB	580TB	2.2PB	250TB	310TB	1.4PB
Rack Units	2U	2U/4U	4U	2U	4U	3U	3U	Dual 3U

iXsystems®

Call iXsystems toll free or visit our website today!

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

intel Xeon® inside™

Dear Readers,

The March issue starts with Rob's column. Don't hesitate to send us your opinions about it. Than, in What's New section you will find Juraj Sipos article on his new release of MaheshaBSD Server. In his article he describes some new features he implemented to it. Now it has WordPress running immediately off a USB memory stick without any installation and configuration procedures.

Next, is Luca's article on reacting to kernel panic. A very detailed, long article containing many codes. If you come across any difficulties when following his steps, just write to him and ask your questions. He will be happy to assist you.

The third part of Rob's series on FreeBSD Programming Primer is demanding this time, bu definitely worth your attention. This time we will look at the code structure, program flow and we will show how to embed CSS and Javascript in our pages. We will also start using SQL to dynamically generate web pages. Speaking shortly, you can learn much out of it.

In section dedicated to security, you will find Michael's last part of Hardening FreeBSD with TrustedBSD and Mandatory Access Controls. It covers the basic configuration of the mac_ifoff, mac_portacl, and MAC LOMAC modules. We hope you enjoyed this series.

Coming soon...

We are preparing for you something really special for April issue – the whole issue will be dedicated to FreeNAS! Don't miss it!

Wish you enjoy the read!
Patrycja Przybylowicz
Editor of BSD Magazine
& BSD Team

MAGAZINE BSD

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

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

Contributing:
Rob Somerville, Luca Ferrari, Michael Shirk, Juraj Sipos

Top Betatesters & Proofreaders:
Ahmed Aneeth, Radjis Mahangoe, Barry Grumbine, Bjørn Michelsen, Paul McMath, Eric Geissinger, Eric De La Cruz Lugo, Imad Soltani, Luca Ferrari, Ewa Duranc

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 Media 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.

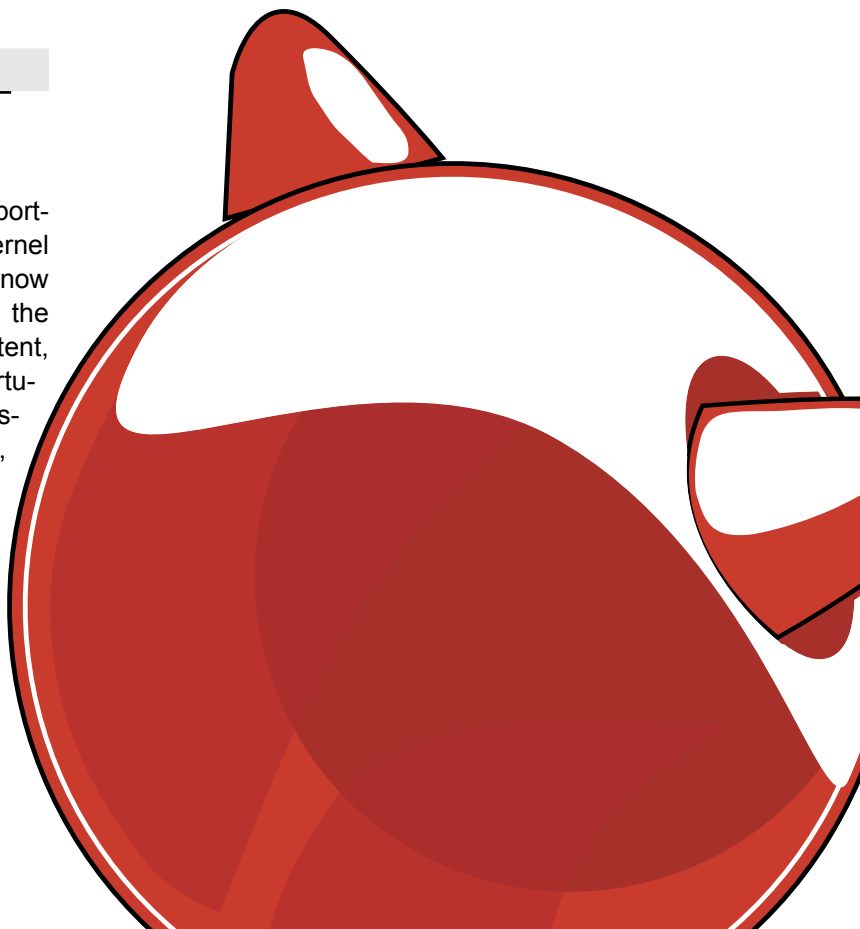
Let's Talk

What's New

How To

Admin

Security



Schrödingers Cat, Tri-states, and the Menace of Metrics

There is continual pressure in our metric driven industry to deliver. The scope triangle defines time, cost and quality as the boundaries that limit any given project. Yet even with this wisdom, projects fail customer expectations, run late or exceed budget. Why?

I believe resolutely that as soon as you start measuring complex systems, the law of Schrödingers cat comes into effect and by the sheer fact of measuring the system, the corresponding result becomes inaccurate. I may be stretching a point here, but bear with me. There are metrics, and there are metrics. I can take a ruler and measure the width of my keyboard (75 mm), and taking a reasoned view I would not expect the plastic of the case to vary in length unless the it was exposed to extreme heat – in which case it would either expand or contract depending on the plastic and thermodynamic properties of the material measured. I have less confidence in the metal tape measure, my ageing eyesight and the shape of they keyboard as it is slightly bevelled at the end and the tape measure is flexible. If I did not measure the width at exactly 90 degrees, the true measurement could be for argument sake, 74-76 mm. So here lies the first law of measurement, in that we have to accept a degree of tolerance or error thereof.

My keyboard is an inanimate object, apart from the life I bestow upon it by supplying it power via USB, pressing the keys and giving it the occasional clean, it has no choice but to be measured. Both my sanity and reality itself would be challenged if the keyboard when presented with the measuring tape decided to move, change shape, or otherwise rebel against measurement.Yet this fact seems to escape some managers when dealing with complex systems. The levels of tolerance and error start becoming so great that the measuring itself becomes the reality, and the facts on the ground statistical aberration.

Take Time Attendance and Management Systems (TAMS). TAMS in itself, no doubt, was designed with noble intentions in mind. By identifying absentee personnel, those that were deliberately trying to defraud the system would be quickly identified, yet as always for the ingenious there is a way to load the dice. Get a colleague to clock in for you, and provided you are both not caught, then cheating the system becomes

quite attractive. However, TAMS has evolved, and using the Bradford Factor, the number of absentees is given a higher negative loading than an sustained period of absence, so you can end up in the scenario where statistically it is more beneficial to have 5 consecutive sick days than 2 or 3 days over a long period. This enters the arena of the law of unintended consequences, as the statistically astute will quickly realise that "throwing a sickie" is more acceptable when the period is extended. In other words, a measurement that was designed to improve efficiency in reality empowers inefficiency. Humans are dynamic and do not like being measured. Schrödingers cat is alive and well in a human resources department near you today.

That illustration aside, take the amount and time and effort it requires to manage and control the quality of the data in TAMS. I have encountered more time management systems in my career than I would wish on my worst enemy, and pretty much all of them are appalling. Inevitably, I end up documenting every nuance of my day that I have to spend time on a Friday afternoon inputting every hour I worked, what I did during the week and why I did it. What is so nihilistic about this whole process is that it is so counter productive – the time spent nurturing TAMS would be so much better spent dealing with that customer, fixing that bug or maybe just staring out the window. And no, I don't get paid overtime. TAMS cannot measure quality of attendance, and by that sheer fact alone it defeats the object of the exercise. And lets not even raise the ethical debate that an employer should trust his employees in the first place (unless of course as a business model they require a level of the dark side in which case the employees must be strictly managed).

The problem with measurement is that like the letter versus the spirit of the law, we can quickly become prisoners of the system. In any given project, there will always be the temptation to counter project creep by getting additional contractors on board. That in itself is not a bad solution, provided that the contractors can hit the ground running, intimately know what is expected and can provide the necessary skill set without stealing time from key personnel on the project. Sadly, so often the team synergy is just not there. In my years of working in IT I have rarely rejoiced at the thought of liaising with 3rd party suppliers, and I have experienced the gamut from the IBM booted and suited to the ripped jeans-wearing anarchist on my travels. The sad fact is that the latter tends to perform better than the former (mainly because they hate project managers and a strong bond develops from the mutual distrust). There is this terrible period of hand holding, trying to balance internal politics versus external commercial reality and in the end this feeling of gloom that as project tech I am going to be left holding the baby. Yet the triangle of time, quality and cost sets the agenda, so I resign myself against every bone in my body that quality will go out the window as time is immutable and that deadline looms. Let the bean counters take the responsibility for the cost, but the project that could have been so much better enters the march of death and I get disappointed that the result could have been excellent rather than just average. Most engineers would get cynical and depressed at this point, but it is important to keep the passion and creativity alive.

This brings us nicely to the philosophy of tri-state. When confronted with the harsh reality of distorted statistics, to maintain our integrity we must adopt a zen like approach and confound those that seek to measure every jot and tittle and kill productivity and creativity in the process. By all means fill in the time sheet, document that code, check that server you know will never fall over and stay within the rules of the system. However, when asked what your secret is return NULL. Even the statisticians have to admit that NULL means no statistical significance exists in a set of given observations. Schrödingers cat is neither here nor there, and there is an ethereal quality about the beast. As engineers who create out of nothing, we must never forget that we bring magic to the black and white of numbers.

ROB SOMERVILLE

Rob Somerville has been passionate about technology since his early teens. A keen advocate of open systems since the mid eighties, he has worked in many corporate sectors including finance, automotive, airlines, government and media in a variety of roles from technical support, system administrator, developer, systems integrator and IT manager. He has moved on from CP/M and nixie tubes but keeps a soldering iron handy just in case.

MaheshaBSD

Server Second Edition Has Been Released…

The Moon is the symbol of Manasa, the Hindu Tantric Serpent Goddess. I chose this Goddess because I believe that not only Unix, but also Tantrism is an aspect of networking. The Full Moon was here on February 25, 2013 – the date when the version 2 of MaheshaBSD Server was released.

What you will learn…

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

What you should know…

- Some knowledge on networking and... Tantrism

The first article about MaheshaBSD Server was published in the 2012 August issue of the BSD Mag (*http://bsdmag.org/magazine/1809-tuning-zfs-on-freebsd*). The story behind MaheshaBSD Server is very well covered in the above-mentioned article. But for those who are impatient – MaheshaBSD Server is primarily an Intranet server. It is a turnkey FTP/WWW Server now also with WordPress running immediately off a USB memory stick without any installation/configuration procedures. It has many advantages, too. Presently, there is not such a thing available in the Open Source world – that is, a pre-configured WordPress Server running off a writable USB memory stick. Particularly WordPress is the news in this release. To set up MySQL and WordPress (WordPress needs MySQL) in FreeBSD is a difficult task for the newbies. Thus, this software has also an educational purpose.

MaheshaBSD Server is built on MaheshaBSD. This means that it has all its functionality (Linux emulation, anonymity, VNC Server, X Window, Text-To-Speech software, etc.).

System Requirements

- Memory (RAM) used by MaheshaBSD Server:

```
mfsroot (root directory, /dev/md0)   - size 54 MB; 6,2 MB free
/tmp                  - size 140 MB
swap                  - size 100 MB
```

Figure 1. *Manasa, the Goddess of cobras. In Hindu mythology, cobra means power and that is what FreeBSD means*

- The minimum memory requirements are 360 MB RAM (look into the "swapme" scripts in `/root/bin` and set up your own swap size).
- A 4 GB USB flash drive minimum (or a USB hard drive of a size larger than 4 GB).

There are not many free online FTP servers today. Most of them (if not all) have restrictions and bypassing them requires payment. With MaheshaBSD Server, you may use any hard disk (NTFS/FAT32 is not a problem either) and set up your own FTP/WordPress server. A beginning cameraman may put a link on his personal website to his home FTP/WordPress server with tons of Gigabytes (videos, MP3's, etc.) of data and share the files with anybody in the world.

Portability is just another advantageous feature – with a straightforward FTP/WWW server (WordPress runs on WWW) you will start your work anywhere in the world in a few seconds with the same files you have had in Brazil after you moved to Canada.

If you configure Samba (it is installed in this software), MaheshaBSD Server can also be used for many things Windows users are accustomed to – for example, you will have a cheap data storage solution accessible via Samba. Many companies need data storage for their industrial cameras, etc. MaheshaBSD Server is a server, so users do not need any special knowledge of Unix. They will work with it the same way they work daily with the Internet.

MaheshaBSD Server has a remote admin – Webmin. Change passwords and administer this software remotely.

Expandability – if you are not satisfied with a particular package (for security reasons, bugs, etc.). In this software, you simply uninstall or upgrade it. As this thing is writable, customize it to your specific needs (install packages of your choice, make your own mail server, etc.). For example, to upgrade WordPress, all you need is to copy a new version of WordPress into `/usr/local/www/wordpress`. To set up your own mail server, use iRedMail for FreeBSD (*http://www.iredmail.org/install_iredmail_on_freebsd.html*). The USB image has 1.4 GB of free space. I can supply larger images upon request. The software is VMware friendly, thus you may run MaheshaBSD Server in Windows as any other application.

Quick Start

Boot. To log in to your WordPress account, put the following entry into your hosts file (`/etc/hosts` in Unix; C:\winnt\system32\drivers\etc in Windows) in Unix or Windows (W2K, XP, Windows Vista, 7 & 8):

```
your_IP manasa
```

like this:

```
192.168.1.200   manasa
```

OR

```
your_ublic_ip   manasa
```

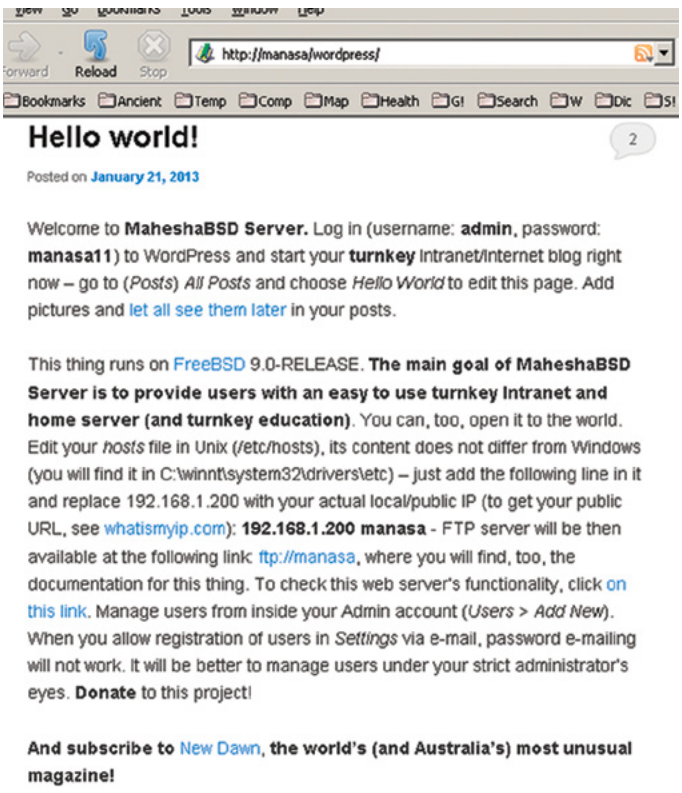


Figure 2. *Log in, edit Hello World and put your company logo wherever you like*

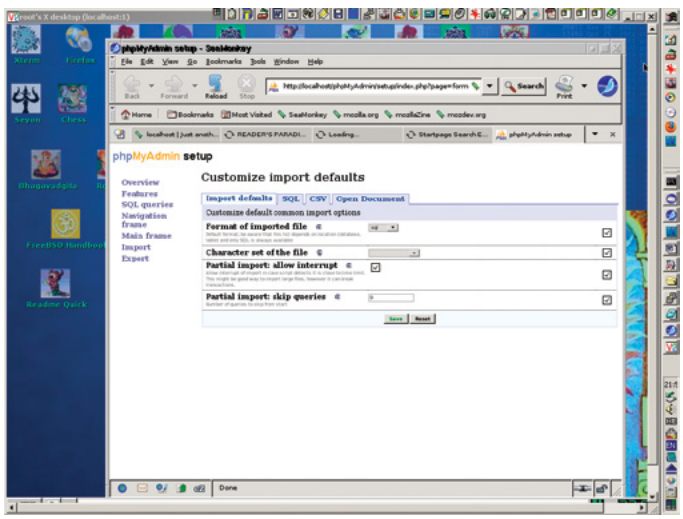


Figure 3. *phpMyAdmin runs both locally and remotely via TightVNC as all MaheshaBSD Server's GUI applications*

Then go to the URL *http://manasa/wordpress* and follow the instructions on the introductory page. Apache server runs on:

- http://manasa
- FTP server runs on:
- ftp://manasa (where you will also find the full documentation for this thing)

The password for root is "freebsdserver". To operate the MaheshaBSD Server's FTP server immediately, log in as user vsftpd (passwords are in `/home/guest5/passes.txt`, so you must first log in as guest5 with password guest6 to fetch all these passwords). Then, copy anything into `/home/vsftpd/ftp` directory (via SFTP – Windows users should use free programs such as WinSCP, etc.). Your files will be immediately available on LAN at the URL *ftp://192.168.1.200* or *ftp://manasa*.

Practical Usage

You may run X Window over the network with free programs such as TightVNC (available also for Windows).All you need is to type vncserver:1 either in the root or guest account on the computer MaheshaBSD Server (or MaheshaBSD) is running on (it will act as a VNC server). The number 1 is the number of the display. If the VNC password is not set up yet, vncserver will prompt you for it (you may change it anytime with vncpasswd). This way you may run phpMyAdmin (a free and open source tool written in PHP intended to handle the administration of MySQL with use of a Web browser) and many other GUI applications remotely (in Microsoft Windows, Linux, Mac, etc.).

To run TightVNC over the Internet (see the picture with Midnight Commander below), you must open the port 5900 (IP Forwarding); the last number is your display number (+1, +2, etc.) – that is, to open MaheshaBSD Server to the world with display:2, the following port must be opened in your router: 5902.

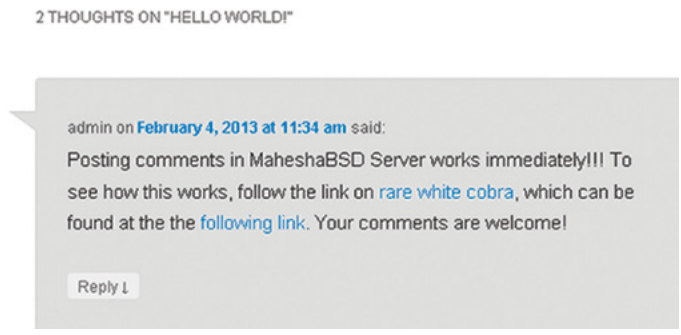


Figure 4. *Leave comments without restrictions – a good way to inform colleagues how far you have progressed with your work*

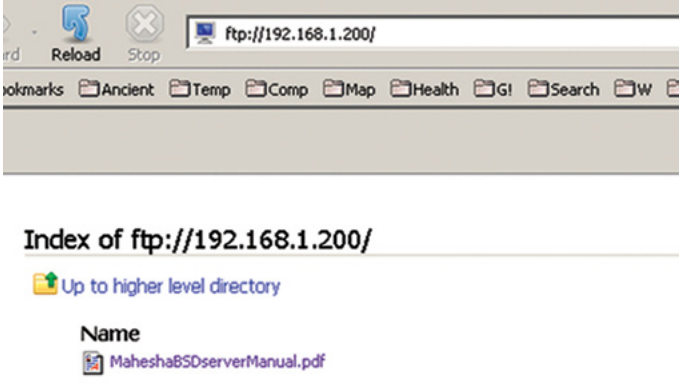


Figure 5. *To have detailed information on how to use and connect to the MaheshaBSD's FTP Server, see the documentation – it is available in the MaheshaBSD Server's anonymous FTP server*

WordPress

Log in as Admin with password manasa11. Go and operate your own Intranet/Internet website. Be public. Have

Figure 6. *It is better to use WordPress with one fixed URL only, as changing the Site Address (URL) all the time your IP changes is complicated*

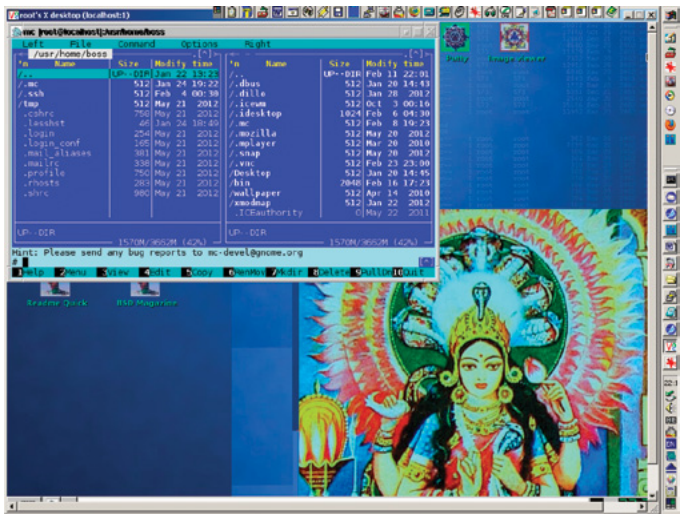


Figure 7. *If you are a beginner, copy files with Midnight Commander (type mc in the command shell, then press F5 to copy a directory/file from the left panel to the right panel)*

your own Internet shop, or the Intranet blog available to all employees.

Before you open WordPress to the world (with IP Forwarding) with MaheshaBSD Server, make sure you change your `/etc/hosts` in Unix, or C:\winnt\system32\drivers\hosts in Windows (W2K, XP, 2003, Windows 7 & 8), otherwise your public WordPress will not be accessible.

If you decide to upgrade WordPress, keep wp-config.php in the `/usr/local/www/wordpress` directory, as it contains the database name used by WordPress. Upgrade/expand anything you think is right for you. WordPress is configurable/expandable up to the maximum.

Goals Of This Project

To provide (users do not need any special knowledge of Unix) small businesses, institutions, households, etc., an immediate and smart FTP storage and a blog solution (WordPress) anywhere in the world.

MaheshaBSD Server has an educational purpose, too. If you study operating systems, it is very valuable to see how FreeBSD works without installing it on your HD (depending on various HD setups, installation sometimes requires repartitioning, a really painful process for many users).

MaheshaBSD Server is a social project. Many people in the Third World need money for education and institutions often fail to provide the valuable infrastructure (software, paying for the development of software, etc.).

Conclusion

I thank *http://www.rootbsd.net* for allowing me to distribute MaheshaBSD and MaheshaBSD Server. The price for this software is 200/150/100/50/25 USD, but only for institutions; MaheshaBSD Server is free for personal use. Institutions may choose any price, thus the software will be valuable also for institutions in very poor countries. Anybody can write and use it for free. Anybody is also welcome to donate. I plan to challenge people to donate money to all BSD projects.

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).

Reacting to Panic

FreeBSD provides a very powerful mechanism for reporting dramatic error conditions that are known as kernel panic.

What you will learn…

- What is a system panic
- How to configure the system to handle crash dumps
- How to get information about what generated a panic

What you should know…

- Basic Shell and Operating System concepts
- Basic C programming language concepts
- How to edit files and compile source code

A kernel panic is when the system does not know how to proceed any further, and instead of making the disks or the memory or any other component inconsistent it raises a panic that is an explicit request for help.

Fortunately, FreeBSD is a secure and stable operating system, and this means that panics are not frequent and, on the other hand, the system is able to start over automatically to provide service continuity. However, knowing what a panic is and how to handle information about it, as well as knowing when, in kernel or device driver development, it does make sense to issue a panic, which is a valuable knowledge for a system administrator.

Let There Be Panic!

The FreeBSD kernel provides a special function, `panic(9)`, that immediately halts the system and prepares it for crash handling. The handling can be either entering the kernel debugger or dumping the memory content to one of the disks and reboot the machine.

The `panic(9)` call is restricted to kernel code, and, therefore, no user program can generate a panic. On the other hand, any piece of kernel code, including loadable modules, can do. This is also the reason for `panic(9)` to exist: every time a piece of kernel code does not know how to proceed without tampering the whole machine (e.g., memory content, disk content, and so on), the system has to panic.

In the following, panics will be discussed in more details, showing to the readers when they can and when they are used within the kernel source code. In order to simulate a real situation, a kernel loadable module will be implemented. So, under specific conditions it will raise a panic. Finally, the inspection with the kernel debugger about what the panic was, will be shown. Readers are suggested to test examples, code listings and in general the panic features on a test machine (for example, a virtual machine), in order to be able to apply their knowledge to production systems in the future.

How Much Panic Can Be?

The FreeBSD code base uses the `panic(9)` method in more than three thousand places in order to report very critical conditions. The idea is that each time the system does not know how to proceed, it generates a panic, so the system will, in the default configuration, reboot and start over. The choice is therefore that instead of becoming inconsystent, the system will try to start over again. This is a reasonable design choice, since panics should come from very strange situations, like an hardware device that is not acting correctly. It is worth noting that panics are not like assertions: `panic(9)` is used to notify a runtime critical situation, while assertions are mainly used to signal a development error. To get an idea consider the `start_init()` routine (`/sys/kern/init_main.c`) which is responsible of starting the init process; what the function

does is essentially to search for a process to execute in a list of paths and to try to run it returning in the case of success. Therefore, as shown in Listing 1, if the `exec(3)` call fails for each path entry, it means that the system has not found an init process to execute, and therefore it cannot continue. This is why the code of Listing 1 "launches" a panic if all the iterations of the *for* cycle fail.

What Happens when the Operating System Goes to Panic?

FreeBSD will not "scream" when entering a panic state, while it is possible some system administrators will do. As strange it sounds, the panic is still a quite "safe" condition, because the system still knows what to do to produce an ordered shutdown. Due to a `panic(9)` call, the system does the following:

- starts an almost ordered (but forced) "shutdown" procedure;
- evaluates if the content of the volatile memory can be "dumped" on persistent storage (more on this later), and if possible stores the content of the memory on disk;
- reboots itself.

At the boot two main different scenarios can happen, the most common of which is to start the system in the usual way, as if an administrator manually turned the machine on. The second possibility is that the kernel enters a "debugging mode" (see the next section), and this is usually interesting for kernel and device drivers developers.

Every time the system boots, an utility command named `savecore(8)` is executed in order to inspect the storage and see if there is a memory dump; if a dump is found the dump is placed into a reachable place in the filesystem. To understand this better, it is important to understand what making a memory dump means.

FreeBSD uses the swap space to keep a memory dump, that means that while the panic is in progress, the kernel will try to write the memory content to a swap area big enough to contain the dump (or will fail if no swap area can be found or can contain the dump). Since the swap area can be used during the boot itself, the usage of the swap area as persistent storage can be risky, and therefore the best the kernel can do is to place the dump at the end of the swap space, so that while booting the machine it will start using the initial swap space (if required) and chances are that swapped data will not overwrite the dump. Moreover, since the recovery of the dump is done in the boot phase, there is a chance that the swap space will not be used entirely (for instance by some daemons), and so the dump can be restored. `savecore(8)` analyzes each swap device configured searching for a consistent dump (foe example, a dump that has not been overwritten by other swapped data during the boot); if a dump is found, the `savecore(8)` command moves such dump out of the insecure swap space to a filesystem configurable directory. After that the boot continues as usual, the swap area can be used entirely and the dump can be inspected with a debugger. It is possible to tune the FreeBSD system specifying exactly where dumps have to be placed and which swap devices must be used to contain a dump: the special variables *dumpdev* and *dumpdir* placed in the `rc.conf` configuration file indicate respectively which swap device has to be used for dumps and to which file system place, the dump has to be written to. Listing 2 shows a configuration example whereas Table 1 summarizes possible values for the above two variables in `/etc/rc.conf`.

Debugging a Panic

On a development machine it could be useful to enter the debugger after a panic is issued, and FreeBSD allows the system administrator to configure the machine to do so. It does suffice to have the debugging options enabled in the current kernel and to enable the *debug.trace_on_panic* syscall to have the system entering the debugger after a panic (see Figure 1).

Listing 1. *The mi_start() procedure*

```
static void
start_init(void *dummy)
{
   …
      for (path = init_path; *path != '\0'; path = next) {
         …
               if ((error = execve(td, &args)) == 0) {
                        mtx_unlock(&Giant);
                        return;
                  }
                  if (error != ENOENT)
                        printf("exec %.*s: error
                     %d\n", (int)(next - path),
                           path, error);
          }
         printf("init: not found in path %s\n", init_path);
         panic("no init");
}
```

Listing 2. *An example of configuration for the dump tunables*

```
dumpdev="/dev/ad0s1b"
dumpdir="/usr/crash"
```

Generating Panics On The Fly

The FreeBSD system allows user administrators to manually crash a system via the *debug.kdb.panic* sysctl; it does suffice to set the above to true (value *1*) to see the system having a panic and rebooting:

```
# sysctl debug.kdb.panic=1
```

While generating panics, using the above sysctl can be a good solution, it does provide the background only for testing the panic-handling configuration of the system and does not allow very much testing and usage of the kernel debugger to discover a few information about what generated the panic itself. Therefore, in the following, a specific kernel module will be implemented to exercise the panic handling.

The panicModule

The `panic(9)` special function is not visible to userland programs, and can be called only from within the kernel context; for this reason in order to use such function to generate on-demand panics it is required to implement a kernel module. An exhausting explanation of kernel module programming is out of the scope of this article, and the module presented here has to be intended only for didactic purposes. This means the module does not strictly follow the kernel code `style(9)` and does not apply optimizations. The module has been implemented and run on FreeBSD i386 versions 8.2 and 9.0-RELEASE, but should work without any changes on other releases as well. The logic behind the module proposed here is quite simple: the module will wrap two file-level system calls, *open* and *mkdir*, and will generate a panic if the path the above system calls are going to be executed contains specific words (for example, "crash"). Given a list of "bad" words defined as *{"panic", "crash", "bsdmag"}* the system will panic if:

- a directory which name (case sensitive) contains at least one of the bad words;
- a file which name (case sensitive) contains at least one of the bad words and the file is going to be written (for example, open in read-write or write-only mode).

While this can seem too complicated for testing panics, it simulates a real module: if the execution context is good (for example, names are clean) the module does its stuff while, if the context is bad (for example, names contain bad words) the module does not know how to proceed and therefore gives up generating a panic.

The module event handler is shown in Listing 3. The module stores the list of bad words, defined as `panic_module_argv_t`) in a global variable named `moduleInitializationData`; such variable will simplify the analysis from behind a system call wrapper. Once the module is loaded (*MOD_LOAD*), the initialization argument of the module is stored in the global variable and the *sysent* entries for the *mkdir* and *open* system calls are changed so that their implementations point to the "panicable" versions. It is worth noting that the module stores in internal variables the old system calls implementation in order to put back in place the original (i.e., not panicing) system calls once it is deactivated (*MOD_UNLOAD*).

Listing 4 shows the implementation of the *mkdir* and *open* wrapping implementations that can generate panics (respectively `panicable_mkdir` and `panicable_open`). The flow is really similar on both functions *panicable_mkdir* and `panicable_open`: the path and the permissions are extracted from the system call arguments and checked against each entry in the list of the bad words (accessed via the `moduleInitializationData` global variable); in the case a match is found the flag `shouldPanic` is set. At the end of both methods if the variable `shouldPanic` is set a

Table 1. *dumpdev and dumpdir possible values*

Variable	Value	Meaning
dumpdev	NO	Disables the crash dump mechanism at all.
	AUTO	Uses the first swap device listed in `/etc/fstab`.
	<swap device>	Uses the specified swap device for the dumping mechanism.
dumpdir	NO	Do not extract a dump from a dumpdev.
	<none>	Uses the default location `/usr/crash`.
	<directory>	Place a found crash from dumpdev into the specified directory.

```
savecore: reboot after panic: Generating a panic from mkdir system call!
Jul 31 09:08:36 bsdmag savecore: reboot after panic: Generating a panic from mkd
ir system call!
savecore: writing core to vmcore.4
Writing crash summary to /usr/crash/core.txt.4.
```

Figure 1. *Entering the debugger from the system console after a panic*

OWNED!

Quick & Easy Security and Compliance Through RedSphere's Secure Hosting Solutions

- Instantly Become PCI Compliant
- We Address Other Compliance and Industry Security Requirements
  - Penetration Testing Services
  - Source Code Security Review and Design
  - Custom Security Services and Solutions

powered by FreeBSD

REDSPHERE©™
Our Promise is Your Peace of Mind

RedSphere Global Security
Call now to speak to a representative
719.924.5266 sales@redsphereglobal.com

www.redsphereglobal.com

Listing 3. *The panic module event handler*

```
static int
panicEventHandler(
                  struct module *st_module,     /*
                    pointer to the module struct */
                  int event,                    /* the
                    event to process for this module */
                  void* argv                    /*
                    custom arguments for the module */
    )
{

  /* Define a variable to handle the exit status (0 on
                    success) */
  int error = 0;

  /*
   *  Define clean (unmodified) system call handlers.
                    These will be used
   *  to revert the original system calls when the
                    module is unloaded.
   */
  sy_call_t *clean_mkdir_system_call = sysent[ SYS_mkdir
                    ].sy_call;;
  sy_call_t *clean_open_system_call  = sysent[ SYS_open
                    ].sy_call;;

  /* which kind of event is happening for this module? */
  switch( event ){

    /* the module is being loaded */
  case MOD_LOAD:
    /* print a message in the logs */
    uprintf( "\nLoading (%d) the module %s ...\n",
                    event, my_name );

    /* get the arguments for the module */
    if( argv != NULL ){
      moduleInitializationData = (panic_module_argv_t*)
                    argv;

      for( int i = 0; i < moduleInitializationData-
                    >count; i++ ){
        uprintf( "\nWill generate a panic on filesystem
                    name containing [%s]",
                  moduleInitializationData->names[i] );
      }

    } /* end of argument for the module */
    else{
      /* set the module arguments to null */
      moduleInitializationData = NULL;
    }

    /* insert the panicable system calls in the system
                    table */
    uprintf( "\nInstalling a panicable mkdir system
                    call...\n" );
    sysent[ SYS_mkdir ].sy_call   = (sy_call_t *)
                    panicable_mkdir;
    uprintf( "\nInstalling a panicable open system
                    call...\n" );
    sysent[ SYS_open ].sy_call    = (sy_call_t *)
                    panicable_open;


    break;

    /* the module is being unloaded */
  case MOD_UNLOAD:

    /* print a message in the logs */
    uprintf( "\nUnloading (%d) the module %s ...\n",
                    event, my_name );

    /* restore the original system call */
    uprintf( "\nRestoring the original system
                    calls...\n" );
    sysent[ SYS_mkdir ].sy_call = clean_mkdir_system_
                    call;
    sysent[ SYS_open  ].sy_call = clean_open_system_
                    call;

    break;

  default:
    /* if here the event is not supported by this module */
    uprintf( "\nEvent/Command %d not supported by module
                    %s\n", event, my_name );
    error = EOPNOTSUPP;
    break;
  }


  /* all done */
  return error;

}
```

Listing 4. *The mkdir and open panicable wrapper implementations*

```
int panicable_mkdir( struct thread *thread,
                     struct mkdir_args *uap
                     )
{
  /* a flag to indicate if should generate a panic or not
                    */
  int shouldPanic = 0;

  /* check if the path for the mkdir call
     is contained into one of the panic paths defined
     at the time of module load */
  for( int i = 0;
       moduleInitializationData != NULL && i <
                    moduleInitializationData->count;
       i++ ){
    if( strstr( uap->path, moduleInitializationData-
                    >names[ i ] ) != NULL ){
      /* the path contains a panicying word! */
      shouldPanic = 1;
      uprintf( "\nThe path [%s] will generate a panic,
                    it contains panic word {%s (index %d)
                    }\n",
               uap->path,
               moduleInitializationData->names[ i ],
               i );
      break;
    }
  }

  /* should we call the standard mkdir or panic? */
  if( ! shouldPanic ){
    /* ok, do a regular call */
    return sys_mkdir( thread, uap );
  }
  else{
    panic( "Generating a panic from mkdir system call!"
                    );
    return shouldPanic; /* should never get here, just
                    to keep quite the compiler */
  }
}

int panicable_open(  struct thread    *thread,
                     struct open_args *uap
                     )
{
  /* a flag to indicate if should generate a panic or not
                    */
  int shouldPanic = 0;

  /* check if the path for the open system call
     is contained into one of the panic paths defined
     at the time of module load, and if so check also
                    the
     opening flags to see if the file is going to be ope
                    ned
     for write
  */
  for( int i = 0;
       moduleInitializationData != NULL && i <
                    moduleInitializationData->count;
       i++ ){
    if( strstr( uap->path, moduleInitializationData-
                    >names[ i ] ) != NULL
        && (  (uap->flags & O_WRONLY) == O_WRONLY
            || (uap->flags & O_RDWR) == O_RDWR )
        ){
      /* the path contains a panicying word, and the file
                    is being opened for writing! */
      shouldPanic = 1;
      uprintf( "\nThe path [%s] will generate a panic,
                    it contains panic word {%s (index
                    %d)} and flags %d\n",
               uap->path,
               moduleInitializationData->names[ i ],
               i,
               uap->flags);
      break;
    }
  }

  /* should we call the standard open or panic? */
  if( ! shouldPanic ){
    /* ok, do a regular call */
    return sys_open( thread, uap );
  }
  else{
    panic( "Generating a panic from open system call!"
                    );
    return shouldPanic; /* should never get here, just
                    to keep quite the compiler */
  }
}
```

Listing 5. *The definition of the module*

```
static char my_name[] = "FreeBSDPanicModule";

static panic_module_argv_t _argv = {
  3,
  { "panic", "crash", "bsdmag" }
};



panic_module_argv_t* moduleInitializationData = NULL;


static moduledata_t panic_module_definition = {
  my_name,                /* module unique name */
  panicEventHandler,      /* event handler */
  &_argv                  /* module arguments */
};



DECLARE_MODULE( panicModuleBin,                          /* binary file name */
                panic_module_definition,                /* module structure name */
                SI_SUB_DRIVERS,                          /* subsystem */
                SI_ORDER_MIDDLE                          /* initialization order */
);
```

Listing 6. *Loading the kernel module into the system*

```
# kldload ./panicModuleBin.ko

Loading (0) the module FreeBSDPanicModule ...
Will generate a panic on filesystem name containing [panic]
Will generate a panic on filesystem name containing [crash]
Will generate a panic on filesystem name containing [bsdmag]
Installing a panicable mkdir system call...

Installing a panicable open system call...
# kldstat
Id Refs Address    Size     Name
 1   20 0xc0400000 bd98b4   kernel
 2    1 0xc0fda000 2820     splash_pcx.ko
 3    1 0xc0fdd000 6434     vesa.ko
 4    1 0xc2df9000 139000   zfs.ko
 5    1 0xc2f32000 3000     opensolaris.ko
 6    1 0xc4743000 3000     daemon_saver.ko
 7    1 0xc2d9c000 2000     panicModuleBin.ko
```

`panic(9)` call is executed and the system crashes, otherwise the original system call is invoked. For the sake of clarity, Listing 5 shows the glue code that defines the module, as well as the initialization data that contains the list of words to match to produce a panic.

The *Makefile* for this module, which code is split among different source files, is the following:

```
KMOD= panicModuleBin
SRCS= panicModule.c panicModuleSyscalls.c
.include <bsd.kmod.mk>
```

and will produce a `panicModuleBin.ko` kernel module (KLD) that can be loaded with the `kldload(8)` command as shown in Listing 6. It is interesting to note that the module informs the administrator of the bad words on which it will generate a crash. In order to generate a panic it does suffice to create a directory or write into a file with a bad-word name as for instance (it does not matter the user identity since the wrapping system calls do not check it):

```
mkdir my_crash_directory
```

and the system will immediately crash showing messages similar to those in Figure 2 on the console. It is worth mentioning that the console shows the exact message the module has passed as argument to the `panic(9)` function. It is also worth noting, as shown in Figure 3,

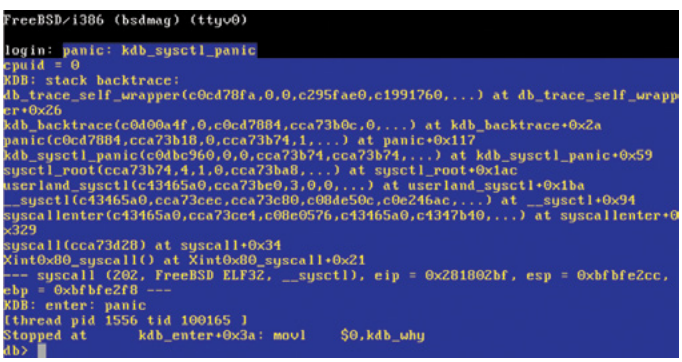


Figure 2. *Console messages immediately after a panic and before the system reboots*

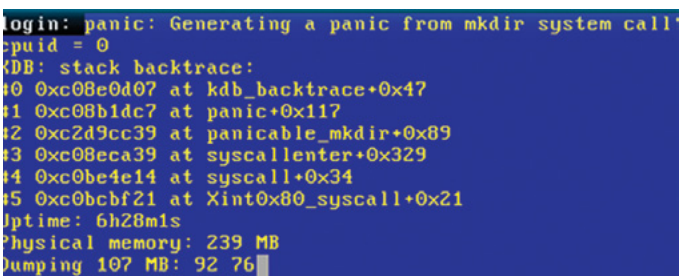


Figure 3. *Kernel messages indicating the saving of a panic dump*

that while booting the system detected a crash-dump and is going to restore it on an accessible file system for further inspection.

In order to use the panic module in a more comfortable way, it is worth installing it in the system so that the module can be found by the loader and the kernel; in particular issuing a *make install* will copy the kernel module object into the default path for modules `/boot/kernel`; it is important to export the symbols for debugging in order to better see what is happening:

```
> make DEBUG_FLAGS=-g3
# make DEBUG_FLAGS=-g3 install
install -o root -g wheel -m 555   panicModuleBin.ko
                    /boot/kernel
install -o root -g wheel -m 555   panicModuleBin.
                    ko.symbols /boot/kernel
kldxref /boot/kernel
```

The Panic Configuration

Now, when it is possible to generate a panic "on-demand", simulating what a real complex kernel module would do, it is time to have a look at how FreeBSD handles panics. As already explained, when the system crash, by means of the `panic(9)` call, FreeBSD dumps the content of the whole memory to a disk, in default to the end of the first swap partition configured in the system. When the system is booted again, the swap partition is checked to see if it contains a previously recordeded dump, and is such case a specific program called `savecore(8)` handles the dump and moves it from the swap to a file-system accessible space, that is to a directory in the system (in default `/var/crash`). As already seen, it is possible to configure either the device/swap partition to use and the final destination of the dump using respectively the variables *dumpdev* and *dumpdir* in `/etc/rc.conf`, as shown in Listing 7.

Once a panic is generated, for instance loading the panic module of the previous section and creating a "bad" directory is as follows:

```
# kldload panicModuleBin.ko
# mkdir go_crash_now
```

the system will automatically reboot and a crash dump will be present in the *dumpdir* location as shown in Listing 8.

As readers can see from Listing 8, there are few numbered files in the *dumpdir* location, so more crashes can coexist in the same place (in the above example the number *0* means no crash have been produced before). The *vmcore.0* file is the biggest one and represents the image of the memory at the time the machine crashed. The *info.0* file is a text file

Listing 7. *A sample configuration of /etc/rc.conf for handling crash dumps*

```
# grep dump /etc/rc.conf
dumpdev="/dev/ad0s1b"
dumpdir="/usr/crash"
# swapinfo
Device          1K-blocks     Used    Avail Capacity
/dev/ad0s1b        610256        0   610256     0%
```

Listing 8. *Acrash dump files after a reboot*

```
# ls -lh /usr/crash/
-rw-r--r--  1 root  wheel     2B Jul 23 15:40 bounds
-rw-------  1 root  wheel   459B Jul 23 15:40 info.0
-rw-------  1 root  wheel    97M Jul 23 15:40 vmcore.0
# more /usr/crash/info.0
Dump header from device /dev/ad0s1b
  Architecture: i386
  Architecture Version: 2
  Dump Length: 101822464B (97 MB)
  Blocksize: 512
  Dumptime: Mon Jul 23 15:39:38 2012
  Hostname: bsdmag
  Magic: FreeBSD Kernel Dump
  Version String: FreeBSD 8.2-RELEASE #2: Mon Jul  9
                  10:59:56 UTC 2012
    root@bsdmag:/usr/obj/usr/src/sys/GENERIC.colors
  Panic String: Generating a panic from mkdir system
                  call!
  Dump Parity: 1825309812
  Bounds: 0
  Dump Status: good
# cat /usr/crash/bounds
1
```

Listing 9. *Starting the kgdb debugger*

```
# kgdb -d /usr/crash/ -n 0 kernel.debug
GNU gdb 6.1.1 [FreeBSD]
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public
                  License, and you are
welcome to change it and/or distribute copies of it
                  under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show
                  warranty" for details.
This GDB was configured as "i386-marcel-freebsd"...

Unread portion of the kernel message buffer:
panic: Generating a panic from mkdir system call!
cpuid = 0
KDB: stack backtrace:
#0 0xc08e0d07 at kdb_backtrace+0x47
#1 0xc08b1dc7 at panic+0x117
#2 0xc4a11c39 at panicable_mkdir+0x89
#3 0xc08eca39 at syscallenter+0x329
#4 0xc0be4e14 at syscall+0x34
#5 0xc0bcbf21 at Xint0x80_syscall+0x21
Uptime: 21m28s
Physical memory: 239 MB
Dumping 97 MB: 82 66 50 34 18 2

Reading symbols from /boot/kernel/splash_pcx.ko...done.
Loaded symbols for /boot/kernel/splash_pcx.ko
Reading symbols from /boot/kernel/vesa.ko...done.
Loaded symbols for /boot/kernel/vesa.ko
Reading symbols from /boot/kernel/zfs.ko...done.
Loaded symbols for /boot/kernel/zfs.ko
Reading symbols from /boot/kernel/opensolaris.ko...done.
Loaded symbols for /boot/kernel/opensolaris.ko
Reading symbols from /boot/kernel/daemon_saver.ko...done.
Loaded symbols for /boot/kernel/daemon_saver.ko
Reading symbols from /boot/kernel/panicModuleBin.ko...
                  Reading symbols from /boot/kernel/
                  panicModuleBin.ko.symbols...done.
done.
Loaded symbols for /boot/kernel/panicModuleBin.ko
#0  doadump () at pcpu.h:231
231             __asm("movl %%fs:0,%0" : "=r" (td));
(kgdb)
```

that contains a summary of the crash generation, such as the time of dump, the kernel ident string, the message of the `panic(9)` call (it is worth noting that it is the same string as the one in the panic module) and the size of the dump (for example, the size of the file *vmcore.0*). The *bounds* file simply contains the name of the next-to-dump set of files: a value of 1 (as of Listing 8) means that the last dump was number *0* and that the next dump-on-crash will be numbered 1.

It is worth noting that the files are owned by the system superuser (*root*) and cannot be read by other users: in fact, the memory dump contains the exact dump of each byte and therefore includes potentially reserved information. Making the file not world-readable is a little security measure to allow only root to inspect such files.

Inspecting a Crash Dump

In order to inspect a crash dump the kgdb(1) utility is required. `kgdb(1)` is the kernel debugger, a modified version of the GNU gdb debugger specialized for analyzing a FreeBSD kernel data. The debugger is interactive and requires to know the core file to inspect or, alternatively, the directory where the crash dumps are and the number of the dump to inspect. As shown in Listing 9, once started, the debugger inspects the core file (*vmcore.0*) and reads the symbols off all modules, including the panic module one. Having loaded the dump data, is now possible to inspect what happened when the system crashed. The symbols for the module are automatically loaded if the module is in the path, otherwise it is possible to use the *add-symbol-file* command of `khd(8)` to add a symbol file from an arbitray path.

It is worth noting that if the kernel has been compiled with debugging symbols, the `kgdb(8)` tool can extract more accurate and useful information about the dump. In particular, the kernel presents a *kernel.debug* directory with all the symbols in the compilation directory for the configuration (usually `/usr/obj/usr/src/sys/<KERNCONF>`); it does suffice to start the `kgdb(8)` command specifying the directory that contains the kernel symbols as follows:

```
cd /usr/obj/usr/src/sys/<KERNCONF>
kgdb -d /usr/crash -n 0 kernel.debug
```

In the following it is assumed that the `kgdb(8)` debugger is started using the kernel symbols; please note that the directory to use is specified in the *info.0* file in the kernel ident string; it is also possible to create a simple shell wrapper that will start `kgdb(8)` with the correct options as shown in Box 2.

The first thing that the `kgdb(8)` utility prints is a kind of "dump header" that corresponds to the message printed out on the console while the system was halting (see Figure 1): the message that has been used as `panic(9)` argument as well as a memory size that will be dumped and the stack trace related to the crash.

Please note

That the data printed in Figure 1 is not effectively the same as the two panics generated are different. However, the concept is that the printed message from `kgdb(9)` is the same as printed out on the console when the panic has been generated.

The first step is getting an idea of what were the last executed instructions, and this can be done inspecting the stack trace, that is unrolling the method calls that terminated with crashing the system. The *backtrace* command can be used to inspect the stack trace as shown in Listing 10. The stack trace is printed out in reverse order, that is the top of the stack represent the most recent call. Therefore, as shown in Listing 10, the call to `panic(9)` (line *#2*) came from a call to `panicable_mkdir()` (line *#3*) which in turn was made by entering a system call via `syscall_enter()` (line *#4*); the last executed instruction was `doadump()` (line *#0*) that halted the system writing down to disk a dump of the whole memory content.

From the above simple information it is possible to get a first idea of what happened: the system crashed due to a voluntary `panic(9)` call, that happened into the `panicable_mkdir` method call. In particular the latter is a system call, since it is on the direct stack of *syscall*.

It is possible to get much more information inspecting what happened within the *sycall*: the `kgdb(8)` allows the selection of a specific frame on the stack and the inspection of the local variables, as shown in Listing 11. The first step requires the selection of the frame number, that is the number reported in the back trace (5 in this example), and then using the *list* command it is possible to see the code executing in such frame: the last line, the 1061, is the line that caused this frame to make another call, and corresponds to the *syscallenter* method call. Using the *info locals* command it is possible to get the dump of the locals variables, that is variable defined into the currently selected stack frame. One variable in particular, *sa*, is important: it is the argument of the actual system call and it provides the data that caused the system call to crash the system. The first thing to note is the *code* attribute, that has a value of 136 corresponding to a *mkdir* system call. Therefore, even without knowing the names or the instructions that caused the panic, it is now possible to state that the system was executing a *mkdir* system call.

Inspecting the stack frame related to the *panicable_mkdir* reveals also more information, and the inspection could have started from there, but it is worth noting that such level

of information could not be always available to the administrator, and this is the reason why the analysis started from the previous frame. As shown in Listing 12, the line 48, that is the last line of code on such executed frame, is the explicit call to the `panic(9)` call. Therefore, it is possible to state that this is the point where the system voluntary crashed. As a quick summary, as shown above, from a crash dump it is possible to get the information about the time and size of

Listing 10. *Examining the stack trace that lead to a crash*

```
(kgdb) backtrace
#0  doadump () at pcpu.h:231
#1  0xc08b1b63 in boot (howto=260) at /usr/src/sys/kern/kern_shutdown.c:419
#2  0xc08b1e00 in panic (fmt=Variable "fmt" is not available.
) at /usr/src/sys/kern/kern_shutdown.c:592
#3  0xc2b0fd59 in panicable_mkdir (thread=Variable "thread" is not available.
) at panicModuleSyscalls.c:48
#4  0xc08eca39 in syscallenter (td=0xc434b870, sa=0xcca2dce4)
    at /usr/src/sys/kern/subr_trap.c:315
#5  0xc0be4e14 in syscall (frame=0xcca2dd28) at /usr/src/sys/i386/i386/trap.c:1061
#6  0xc0bcbf21 in Xint0x80_syscall () at /usr/src/sys/i386/i386/exception.s:264
#7  0x00000033 in ?? ()
```

Listing 11. *Exploring the syscall stack frame*

```
(kgdb) frame 5
#5  0xc0be4e14 in syscall (frame=0xcca27d28) at /usr/src/sys/i386/i386/trap.c:1061
1061            error = syscallenter(td, &sa);
(kgdb) list
1056            orig_tf_eflags = frame->tf_eflags;
1057
1058            td = curthread;
1059            td->td_frame = frame;
1060
1061            error = syscallenter(td, &sa);
1062
1063            /*
1064             * Traced syscall.
1065             */
(kgdb) info locals
td = (struct thread *) 0xc439d000
sa = {code = 136, callp = 0xc0d8e3c0, args = {-1077940878, 511, 0, -1077942756, 0, 0, 0,
    -1002844160}, narg = 2}
orig_tf_eflags = 582
error = -1002844160
ksi = {ksi_link = {tqe_next = 0xcca27d28, tqe_prev = 0xc0dfcf80}, ksi_info = {
    si_signo = 3, si_errno = -998280536, si_code = 0, si_pid = 3, si_uid = 4,
    si_status = 12, si_addr = 0x2816c42c, si_value = {sival_int = 672580652,
      sival_ptr = 0x2816c42c, sigval_int = 672580652, sigval_ptr = 0x2816c42c},
    _reason = {_fault = {_trapno = 519}, _timer = {_timerid = 519,
        _overrun = -861766420}, _mesgq = {_mqd = 519}, _poll = {_band = 519},
      __spare__ = {__spare1__ = 519, __spare2__ = {-861766420, -1064589813, -1002844160,
          -998280536, -861766420, -1064385552, -1002844160}}}}, ksi_flags = -1002844160,
  ksi_sigq = 0xcca27cec}
```

the dump (for example, from the *info.0* file), and thru the debugger `kgdb(8)` it was possible to find out that the last system call executed was a *mkdir* (number 136) and that such syscall, that resolved into a *panicable_mkdir* autonomously generated a dump calling `panic(9)` at line 48.

It is now possible, if the compilation allows to (see Box 1), to inspect also the local variables for the system call stack frame; the procedure is similar to that presented before and is shown in Listing 13. Unluckily, while it is quite simple to see locals, heap based variables cannot be inspected since their value could be in user space and not in the kernel address range. This is the case of the *uap->path* variable, which is a pointer to the memory address `0xbfbfee16` that is under the 3GB memory limit (32 bit architecture) `0xc000000` (address over this limit are limited to kernel usage). In this case it is possible to do a change to the `panicable_mkdir` system call in order to either copy the variable on the stack or into a kernel memory address. Listing 14 shows both the approaches. Listing 15 shows the result of such change: the string `go_crash_now` is now correctly reported in both the local stack variable and the kernel memory pointer. Therefore, in this lucky case it is possible to state that the crash was generated by a *mkdir* system call with `go_crash_now` as main argument.

In the unlucky case there is no way to access the content of the *uap->path* variable, a more deep inspection has to be done. Having the source code and the debugging symbols is very helpful: in fact the administrator/developer can inspect more code, for instance what happened before the generation of the panic, and try to understand what happened and what brought the system to panic. So far, the administrator knows that a *mkdir* system call has been invoked, and that the `shouldPnanic` variable has been set, so to generate a panic (see Listing 12). But where was such variable set? The *list* command can help: the debugger can show previous lines on the selected frame, so that giving *list -10* will print the code of the system call implementation 10 lines backward at time. Therefore, the code that happens is that of Listing 16, that reveals that the `shouldPanic` flag is set depending on the values contained in a global variable `moduleInitializationData`. It is then possible to manually inspect such variable to get more clues; as shown in Listing 16 the dump reveals all the bad words used to panic the system. Therefore, having the source code information in this case helps to the point that a *mkdir* system call executed with one of the bad words contained in `moduleInitializationData` crashed the system.

The Dumping Mechanism: a Few Low Level Details

It is interesting to navigate the operating system source code in order to see what is effectively done when a dump needs to be written to persistent storage. In the following, a quick explanation of the main routines involved in the dump mechanism is presented.

The dump and shutdown sequence

The *kern_shutdown.c* file defines the `doadump()` function that is responsible for starting the dump of the memory. The `doadump()` function essentially checks that a dump device is configured (otherwise it simply returns) and then performs the saving of the context registers and starts the real workhorse for dumping, that is the `dumpsys(..)` function as shown in Listing 17. The *dumper* object is defined as `dumper_info` struct and one of its most important field is the *dumper* one, that is a pointer to the function that is going to do the real dump of the data.

The `dumpsys(..)` function is a low level function and is therefore machine dependent; as an example consider the implementation for the Intel x86 architecture contained in *sys/x86/x86/dump_machdep.c* and which snippet is shown in Listing 17. A detailed explanation of such low level function is out of the scope of this article, so a quick walk through will be presented. Initially the `dumpsys(..)` function checks if the media pointed by the *di* variable contains enough space to handle the whole dump (which size is expressed by the *dumpsize* variable) and the headers of the dump. Supposing the media has enough space, the *dumplo* variable is initialized with the location offset of where the dump must be placed. Please note that the dump position is computed starting from the end of the media and coming to its start, so that the dump will result at the end of the media. This is due to the fact that the media is effectively a swap used storage, and since the system will require swap space to boot, placing the dump at the beginning of the swap will make the boot process to overwrite it. Placing the dump at the end of the swap space give more chances that the boot process will not consume so much swap area to overwrite the data that is at the end. The remaining of the `dumpsys(..)` performs a set of `dump_write(..)` calls to write the header and the whole memory content. The call to the `foreach_chunk(..)` method performs the actual volatile memory dump. The `foreach_chunk(..)` in turns call the `cb_dumpdata(..)` (see Listing 17) that essentially does the following:

- computes the number of pages to dump and store it in *pgs* variable;
- computes how many pages can be dumped in a single I/O operation on the media (*maxdumppgs*);
- executes a *while* loop on the number of pages to dump *pgs* and for each iteration computes the number of pages that will be dumped (*chunk*) and the size of the dump (*sz*);

Listing 12. *Information from the panicable_mkdir stack frame*

```
(kgdb) frame 3
#3  0xc2b0fd59 in panicable_mkdir (thread=Variable
                    "thread" is not available.
) at panicModuleSyscalls.c:48
48          panic( "Generating a panic from mkdir system
                    call!" );
(kgdb) list
43        if( ! shouldPanic ){
44          /* ok, do a regular call */
45          return sys_mkdir( thread, uap );
46        }
47        else{
48          panic( "Generating a panic from mkdir system
                    call!" );
49          return shouldPanic; /* should never get here,
                    just to keep quite the compiler */
50        }
```

Listing 13. *Inspecting local variables for the panicable_mkdir system call*

```
(kgdb) info locals
shouldPanic = 1
(kgdb) set print pretty
(kgdb) print *uap
$1 = {
  path_l_ = 0xcdba3cec "\026□□\001",
  path = 0xbfbfee16 <Address 0xbfbfee16 out of bounds>,
  path_r_ = 0xcdba3cf0 "□\001",
  mode_l_ = 0xcdba3cf0 "□\001",
  mode = 511,
  mode_r_ = 0xcdba3cf4 ""
}
(kgdb) print * 0xbfbfee16
Cannot access memory at address 0xbfbfee16
```

Listing 14. *The first part of the panicable_mkdir implementation that can copy the path into a local variable or a kernel memory address*

```
#ifdef _MALLOC_ARGS_
static MALLOC_DEFINE( M_PANIC_MEMORY, "mkdir-path",
                    "Argument to the mkdir system call" );
#endif //_MALLOC_ARGS_

int panicable_mkdir( struct thread *thread,
                     struct mkdir_args *uap
                     )
{
  /* a flag to indicate if should generate a panic or not */
  int shouldPanic = 0;

#ifdef _LOCAL_PATH_

  char localPath[ 255 ];

  /* zero fill the memory */
  for( int i = 0; i < 255; i++ )
    localPath[ i ] = '\0';

  /* copy the mkdir path to a local variable, so it will
                     be available
     when using the debugger */
  strcpy( localPath, uap->path );
#endif /* _LOCAL_PATH_ */
#ifdef _MALLOC_ARGS_
  char *kPath = malloc( strlen( uap->path ) + 1 , M_
                     PANIC_MEMORY, M_NOWAIT );


  // copy the uap into the kernel version
  copyinstr( uap->path, kPath, strlen( uap->path ), NULL );
#endif //_MALLOC_ARGS_
```

Listing 15. *Inspecting the local copy of the path.*

```
(kgdb) info locals
(kgdb) info locals
shouldPanic = 1
localPath = "go_crash_now", '\0' <repeats 242 times>
localPathPointer = 0xbfbfee16 <Address 0xbfbfee16 out of
                     bounds>
kPath = 0xc43e5160 "go_crash_now"
```

Listing 16. *Inspecting the moduleInitializationData global variable*

```
(kgdb) print moduleInitializationData
$17 = (panic_module_argv_t *) 0xc4a3606c
(kgdb) print *moduleInitializationData
$18 = {
  count = 3,
  names = 0xc4a36070
}
(kgdb) print (*moduleInitializationData).names[0]
$19 = 0xc4a34df5 "panic"
(kgdb) print (*moduleInitializationData).names[1]
$20 = 0xc4a34dfb "crash"
(kgdb) print (*moduleInitializationData).names[2]
$21 = 0xc4a34e01 "bsdmag"
```

Great Specials

On FreeBSD® & PC-BSD® Merchandise

Give us a call & ask about our
SOFTWARE BUNDLES
1.925.240.6652

$39.95
FreeBSD 9.1 Jewel Case CD Set
or FreeBSD 9.1 DVD

$29.95
PC-BSD 9.1 DVD

$49.95 Save a BUNDLE!
The PC-BSD 9.0 Users Handbook
PC-BSD 9.1 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 9.1 CD or DVD set
FreeBSD Toolkit DVD

Stylish Dress Attire
Look Your Professional Best

Comfy Apparel
Stay Warm in Zip Ups & Pullovers

T-Shirts
Lots of Styles to Choose From

FreeBSD 9.1 Jewel Case CD/DVD $39.95

CD Set Contains:

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

FreeBSD 9.0 CD $39.95
FreeBSD 9.0 DVD $39.95

FreeBSD Subscriptions

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

FreeBSD Subscription, start with CD 9.1 ..... $29.95
FreeBSD Subscription, start with DVD 9.1 ..... $29.95
FreeBSD Subscription, start with CD 9.0 ..... $29.95
FreeBSD Subscription, start with DVD 9.0 ..... $29.95

PC-BSD 9.1 DVD (Isotope Edition)

PC-BSD 9.1 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 9.1 ..... $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

Listing 17. *The doadump() kernel function*

```
static void
doadump(void)
{
        if (dumper.dumper == NULL) {
                printf("Cannot dump. Device not defined
                    or unavailable.\n");
                return;
        }

        savectx(&dumppcb);
        dumptid = curthread->td_tid;
        dumping++;
        dumpsys(&dumper);
        dumping--;
}

static struct dumperinfo dumper;

// from sys/sys/conf.h
struct dumperinfo {
        dumper_t *dumper;       /* Dumping function. */
        void    *priv;          /* Private parts. */
        u_int   blocksize;      /* Size of block in
                                   bytes. */
        u_int   maxiosize;      /* Max size allowed for
                                   an individual I/O */
        off_t   mediaoffset;    /* Initial offset in
                                   bytes. */
        off_t   mediasize;      /* Space available in
                                   bytes. */
};
```

Listing 18. *An extract of the dumpsys(..) function for the x86 architecture*

```
void
dumpsys(struct dumperinfo *di)
{
        Elf_Ehdr ehdr;
        uint64_t dumpsize;
        off_t hdrgap;
        size_t hdrsz;
        int error;
…
        if (di->mediasize < SIZEOF_METADATA + dumpsize +
                    sizeof(kdh) * 2) {
                error = ENOSPC;
                goto fail;
        }

        dumplo = di->mediaoffset + di->mediasize -
                    dumpsize;
        dumplo -= sizeof(kdh) * 2;
…
/* Dump memory chunks (updates dumplo) */
        error = foreach_chunk(cb_dumpdata, di);
        if (error < 0)
                goto fail;
…
        dump_write(di, NULL, 0, 0, 0);
        printf("\nDump complete\n");
        return;
}

static int
cb_dumpdata(struct md_pa *mdp, int seqnr, void *arg)
{
        struct dumperinfo *di = (struct dumperinfo*)arg;
        vm_paddr_t a, pa;
        void *va;
        uint64_t pgs;
…
        pgs = mdp->md_size / PAGE_SIZE;
        pa = mdp->md_start;
        maxdumppgs = min(di->maxiosize / PAGE_SIZE,
                    MAXDUMPPGS);
        if (maxdumppgs == 0)    /* seatbelt */
                maxdumppgs = 1;
…
        while (pgs) {
                chunk = pgs;
                if (chunk > maxdumppgs)
                        chunk = maxdumppgs;
                sz = chunk << PAGE_SHIFT;
                counter += sz;

                for (i = 0; i < chunk; i++) {
                        a = pa + i * PAGE_SIZE;
                        va = pmap_kenter_
                    temporary(trunc_page(a), i);
                }
                error = dump_write(di, va, 0, dumplo,
                    sz);
                if (error)
                        break;
                dumplo += sz;
                pgs -= chunk;
                pa += sz;
...
```

- the virtual address *va* of the pages to be dumped is extracted;
- the `dump _ write(..)` method is called to dump the *va* virtual address on the *di* dump media for the computed size *sz* starting at the offset *dumplo*;
- all offset and size counters are increased.

The `dump _ wtite(..)` method is only a wrapper for the `dumperinfo.dumper(..)` method that is responsible for the physical I/O dump of the data. Usually the *dumper* method is associated to the *d_dump* field of a *GEOM* structure, and as an example the dump method for the ATA disk drives is shown in Listing 19 (extracted from `dev/ata/ata-disk.c`). The method simply prepares a *bio* structure (Block I/O) that will reference the virtual address passed as argument (for example, the page with the data to dump), will write on the physical storage no more data than the length passed as argument using the specified offset. Once the *bio* structure is ready with the above data, it is passed to the GEOM layer *startegy* routine for being processed and enqueued for the physical write.

As a final note, please take into account that a dump header is placed at both boundaries of the dump itself, that is at the beginning and at the end. This means that on the very last sector of the dump media there will be a dump header, and this header is used during the restoration mechanism (see next section) in order to validate and get information about the size of the dump itself.

The Restoring Mechanism

As already stated, the `savecore(8)` program is responsible for extracting a core dump and placing it in a safe position of the file system. The `savecore(8)` command is a userland program that has to be run with administrator privileges, and

Box 1. Debugging Symbols

When compiling a module like the panic one, or every time it is worth debugging a piece of code, the compilation should include the DEBUG_FLAGS, that tell the compiler to generate the symbol table that the debugger can understand. It must also be took into account that, by default, a module compilation provides optimizations, that is the compiler flags -O2 are used. While this is fine for production systems, for development/debugging system this is suboptimal, since such compiler optimization can produce code that is not "mapped" back to source and therefore is harder to debug. For this reason, it is better to compile a development system with optimization turned off, that is with explicit flag -O0. In order to have also the kernel symbols, the kernel itself must be compiled in debug mode, that is the kernel configuration file must have the option:

```
makeoptions DEBUG=-g
makeoptions DEBUG=-g
```

essentially it performs file operations using the well defined system calls for accessing data on the swap device, and using the C file routines to access the filesystem. The more interesting routine in the program is the `DoRegularFile(..)` which is sketched in Listing 20 and that is called with the file descriptor of the swap device and the name of the destination file (plus other parameters like the size of the dump). As readers can see, the function `DoRegularFile(..)` performs a loop for the size of the dump, and at each step reads a *wl* size of data from the dump device, and writes out the read data to the *fp* file pointer. What is not shown in Listing 20 is that *fp* is a regular file pointer or a compressed one (for example, `fopen(3)` or *zopen(3)*) and in the case the saved core has not to be compressed (for example, *compressed = 0*) all dump blocks that are zero filled are skipped to not waste space. Before the `DoRegularFile(..)` routine can be called, the `savecore(8)` program has to do several checks and preparation: as shown in Listing 21 it must move the file offset of the dump device to the where the dump is (remember that a dump is placed at the end of the swap device), then a check on the dump headers is performed and dump information is printed via `printheader(..)` (see Listing 8 for an example), as well as the panic string if the dump comes from a panic (see Listing 9 for an example). Then a reasonable check on the disk space is performed and the real work is done via the above `DoRegularFile(..)` routine.

Listing 19. *The dump routine for an ATA disk*

```
static int
ad_dump(void *arg, void *virtual, vm_offset_t physical,
        off_t offset, size_t length)
{
…
bzero(&bp, sizeof(struct bio));
    bp.bio_disk = dp;
    bp.bio_pblkno = offset / DEV_BSIZE;
    bp.bio_bcount = length;
    bp.bio_data = virtual;
    bp.bio_cmd = BIO_WRITE;
    ad_strategy(&bp);
    return bp.bio_error;
}
```

Listing 20. *The DoRegulaFile extraction routine*

```
static int
DoRegularFile(int fd, off_t dumpsize, char *buf, const
                    char *device,
    const char *filename, FILE *fp)
{
        int he, hs, nr, nw, wl;
        off_t dmpcnt;

        dmpcnt = 0;
        he = 0;
        while (dumpsize > 0) {
                wl = BUFFERSIZE;
                if (wl > dumpsize)
                        wl = dumpsize;
                nr = read(fd, buf, wl);
                ...
                if (compress) {
                        nw = fwrite(buf, 1, wl, fp);
```

Listing 21. *Preliminars steps of the savecore command*

```
        firsthd = lasthd - dumpsize - sizeof kdhf;
        lseek(fd, firsthd, SEEK_SET);
        error = read(fd, &kdhf, sizeof kdhf);
        if (error != sizeof kdhf) {
                syslog(LOG_ERR,
                    "error reading first dump header at
                    offset %lld in %s: %m",
                    (long long)firsthd, device);
                nerr++;
                goto closefd;
        }
...
         printheader(stdout, &kdhf, device, bounds, -1);
…
       if (kdhl.panicstring[0])
                syslog(LOG_ALERT, "reboot after panic:
                    %s", kdhl.panicstring);
        else
                syslog(LOG_ALERT, "reboot");

        if (verbose)
                printf("Checking for available free
                    space\n");
        if (!check_space(savedir, dumpsize)) {
                nerr++;
                goto closefd;
        }
…
if (DoRegularFile(fd, dumpsize, buf, device, filename,
                    fp)
                    < 0)
                        goto closeall;
…
```

Conclusions

This article has presented the FreeBSD kernel panic mechanism and when the `panic(9)` call should be used to report a very dramatic condition within the kernel or a loadable module. The article also shown how to handle a crash dump and to inspect it, assuming a quite luckily condition where the administrator or the developer has the source code and can inspect it with the use of the kernel debugger. For this purpose a kernel module that simulates a real scenario, where a panic is generated due to abnormal conditions, has been implemented to both allow the user to experiment an "on-demand" panic situation and to show how to use the `panic(9)` call. Finally, a few details about how the dumping and restoring of the dump data are performed by the system has been shown.

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 login to non-Unix systems. He can be reached on-line at http://fluca1978.blogspot.com.

Box 2. A simple shell script to launch the kernel debugger

The following script is a simple wrapper that accepts the dump number and the dump directory (or set them to zero and *dumpdir* – from `/etc/rc.conf` – by default) and launches the kernel debugger (after having done a few other checks about the file existence, the user being root and so on).

```
#!/bin/sh

# A Kernel Debugger launching script.
# Usage:
# start_kgdb.sh [dump-number  [dump-dir] ]
#
# example:
# start_kgdb.sh 0 /usr/crash
#
# If not specified the dump number is assumed 0, the dump directory
# is extracted from /etc/rc.conf.


# check I'm root
ID=`id -u"`
if [ $ID -ne 0 ]
then
    echo "Sorry, you need to be root to execute this script!"
    exit
fi



# first argument is the dump number or zero by default
if [ $# -gt 0 ]
then
    DUMP_NUMBER=$1
else
    DUMP_NUMBER=0
fi

# second argument is the dump directory or
# the directory defined in the rc.conf file
if [ $# -gt 1 ]
then
    DUMP_DIR=$2
else
    . /etc/rc.conf
    if [ ! -z "$dumpdir" ]
    then
        DUMP_DIR=$dumpdir
    else
        DUMP_DIR=/var/crash
    fi

fi

echo "Debugging for dump number $DUMP_NUMBER in $DUMP_DIR"

# build files for core and info
DUMP_FILE_CORE=${DUMP_DIR}/vmcore.${DUMP_NUMBER}
DUMP_FILE_INFO=${DUMP_DIR}/info.${DUMP_NUMBER}

# check that files exist
if [ ! -f $DUMP_FILE_CORE -o ! -f $DUMP_FILE_INFO ]
then
    echo "Cannot find dump core/info files!"
    echo "I was looking for $DUMP_FILE_CORE and $DUMP_FILE_INFO"
    exit
fi



# get the kernel directory and the debug file
HOSTNAME=`hostname`
HOSTNAME="@"${HOSTNAME}
KERNEL_D=`grep "${HOSTNAME}" ${DUMP_FILE_INFO} | awk -F ":" '
                    {print $2;}'`
KERNEL_D=${KERNEL_D}/kernel.debug
if [ !  -f $KERNEL_D ]
then
    echo "Cannot find kernel debug symbols!"
    echo "Kernel debug info was $KERNEL_D"
    KERNEL_D=""
fi

# ok start the debugger
kgdb -d $DUMP_DIR -n $DUMP_NUMBER $KERNEL_D
```

On The Web

- Source code of the kernel module for generating on-demand panics: *https://github.com/fluca1978/fluca1978-coding-bits/tree/master/teaching-stuff/FreeBSD/panic_module*
- FreeBSD Documentation on Kernel Debugging: *http://www.freebsd.org/doc/en/books/developers-handbook/kerneldebug.html*

FreeBSD Programming Primer – Part 3

In the third part of our series on programming, we will look at code structure, program flow and how to embed CSS and Javascript in our pages. We will also start using SQL to dynamically generate web pages.

What you will learn…

- How to to configure a development environment and write HTML, CSS, PHP and SQL code

What you should know…

- BSD and general PC administration skills

Before we start coding in earnest, we will look at the basic construction of our programming language (PHP), the directory and functional structure of our CMS and how this all fits together.

Our CMS will be designed to be as extensible as possible, and will follow the design as detailed in Figure 1. Pages will be stored in the MySQL database, merged with the header, templates, CSS and Javascript and returned to the client browser. This will allow us to separate design from content efficiently.

Part 1. PHP Fundamentals

Any language – both verbal and programming – comprises of separate elements that fit together in a logical structure that communicates meaning to the recipient. For language to be effective, rules are strictly defined, not only to preserve efficiency but to prevent misunderstanding. For example, a written sentence will comprise of nouns, verbs and adjectives – likewise computer code will consist of variables, expressions and control structures. The main functional difference between a human language such as English and a programming language is flexibility and interpretation – as humans we are adaptable enough to interpret a missing full stop or misspelled word correctly, whereas a computer will fail miserably.

Here will will look at some of the the basic building blocks of PHP.

The following code examples can be created in the examples directory using your favorite editor (in this case I am using VI). Login to the webserver using SSH or from the console, switch to the DEV account from root and create the files:

```
dev# su
dev# su dev
dev# cd /home/dev/data/
dev# mkdir examples
```

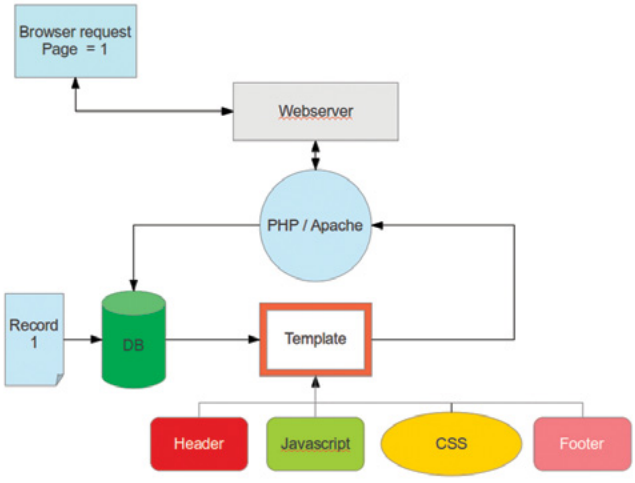


Figure 1. *CMS design*

```
dev# cd examples
dev# vi example1.php
```

The example can then be run from: *http://yourserveripaddress/examples/example1.php* (see Figure 2).

You do not need to run the examples to develop a working CMS, but they are useful to experiment with. Change values and experiment.

PHP tags

All PHP code requires an opening `<?php` tag. The closing `?>` tag is only necessary when combining PHP with Javascript, HTML etc.

Comments

In PHP comments are denoted with `//`. A block of comments can also be with a block using `/* …. */`.

Expressions

To quote the PHP website "Expressions are the most important building stones of PHP. In PHP, almost anything you write is an expression. The simplest yet most accurate way to define an expression is "anything that has a value".

For instance, if the server has 3 disk drives this could be written as:

```
<?php
$disk_drives = 3;
```

Constants

A constant is useful where the the value will not change across the execution of the script. Unlike variables, constants are accessible within function calls and throughout the script, whereas variables may be local only to that particular function. This aids program readability.

Figure 2. *example1.php running via a browser*

The circumference of a circle is calculated by multiplying the diameter by PI (3.14). As PI is a constant and will not change, we can define PI as a constant. See Listing 1.

Variables

A variable holds the result of an expression or statement. As the name implies, the value of the variable will change over the life of the program. The variable name is defined on the left hand side of the = sign and the value of the variable is defined on the right hand side see Listing 2.

Data types

Data types, as the name suggests, define what type of data we can hold in a variable or constant. The basic types are booleans, integers, floats, strings, and arrays.

Booleans

A boolean is used where a dual state is useful. A boolean has one of two values, TRUE or FALSE. For instance, we can define the constant DEBUG and act accordingly depending on how DEBUG evaluated by the "if" control structure: see Listing 3.

Listing 1. *example1.php*

```
<?php

/*
 * example1.php
 * Constants
 * Define PI as the constant 3.14 and output the result.
 *
 */

define("PI", 3.14);
echo PI;
```

Listing 2. *example2.php*

```
<?php

/*
 * example2.php
 * Variables
 * Define circumference as the variable $circumference
                    with the value
 * 12.775 and output the result
 *
 */
$circumference = 12.775;
echo $circumference;
```

Integers

An integer is a whole number of the set `Z = {..., -2, -1, 0, 1, 2, …}`. As the maximum and minimum value is platform independent, we can access this value via the constant `PHP_INT_MAX`. If PHP encounters a number larger than `PHP_INT_MAX`, the number will be converted to a float. See Listing 4.

Floats

The maximum size of a floating point number, like Integers, is platform dependent. However, all sorts of rounding and logic errors can be introduced into code with floats as they behave differently from integers so particular care should be used. For instance, 1/3 is equal to 0.33333 with 3 recurring to infinity, but as we have limited space it is impossible to represent this fully. If accuracy is critical, various libraries are available to improve float functionality. See Listing 5.

Strings

A PHP string can contain up to 2Gb of characters. A string is limited to 256 ASCII characters, and does not internally store strings in Unicode format unlike some other languages.

A string can be surrounded either by single or double quotes. If surrounded by double quotes, PHP will endeavor to evaluate any variables contained within. A single quoted string is called a string literal as the string is interpreted literally rather than expanding any variables contained within it. Like the Vi versus Emacs discussion, the use of single or double quotes is very much a question of what you want to achieve. While single quotes may be considered quicker than double quotes, other coding factors have a greater impact on speed. See Listing 6.

Arrays

An array is a list with a key and value pair. For instance, we can list the major BSD distributions as an array variable, rather than multiple separate variables. We can then perform operations on the list by looping through the key /value pairs.

Arrays are useful where we have to keep records in sync. For instance if the records from a database table were dumped into an array it would be easy to manage using the record ID as key. If separate variables were used, it would be difficult to manage and the potential for errors would be great. Arrays do not need to have sequential keys, indeed PHP supports the mixing of numeric and string values as keys within arrays. You can even define another array as the value of an array

Listing 3. *example3.php*

```
<?php

/*
 * example3.php
 * Booleans
 * Define DEBUG as a boolean, and print our status.
 * Change TRUE to FALSE to change the output message.
 *
 */

define("DEBUG", TRUE);
if (DEBUG) {
    echo 'We are in Debug mode';
} else {
    echo 'We are not in Debug mode';
}
```

Listing 4. *example4.php*

```
<?php

/*
 * example4.php
 * Integers
 * Check the maximum integer size available on your
                    platform.
 * For a 32 bit system this will be 2147483647.
 *
 */

echo PHP_INT_MAX;
```

Listing 5. *example5.php*

```
<?php

/*
 * example5.php
 * Floats
 * Calculate PI as a float using the more accurate
                    formula 22 / 7.
 * This should return 3.1428571428571.
 *
 */

$pi = 22 / 7;
echo $pi;
```

The first array example is the traditional PHP method for defining arrays, the second version onwards is available in > PHP 5.4. See Listing 7.

Operators

Operators are used to compare values (Comparison / Logical) or change values (Arithmetic / Assignment / Increment / Decrement / String). Care has to be taken with operator precedence, as certain operators will fire first. For example, the multiplication operator * takes precedence over the addition operator + unless the addition portion of the equation is wrapped in brackets: see Listing 8. See Table 1 for the full list of of the most common operators.

Functions

Functions are small blocks of code that can act as effectively as a "black box" to the code that is calling it. As functions can be called repeatedly from anywhere within code – and if written properly – will provide a consistent result. Functions can act independently of the code that is calling them, or can return a result that can be manipulated by the main body of the program. An important point to realize is that variables defined inside a function are generally out of scope of the main body – that is to say $a in the main body of a program cannot be accessed by the function unless it is either passed as a parameter or accessed via some other method (PHP has a rich library of internal functions, if

Listing 6. *example6.php*

```
<?php

/*
 * example6.php
 * Strings
 * Demonstration of the PHP string type.
 * Note that the last line is functionally identical to
                    the previous
 * line and we are separating each line with a HTML <br />.
 *
 */

define("BR", '<br />');
$pi = 22 / 7;
echo 'The value of PI is: $pi' . BR;
echo 'The value of PI is: ' . $pi . BR;
echo "The value of PI is: $pi" . BR;
```

Listing 7. *example7.php*

```
<?php

/*
 * example7.php
 * Arrays
 * All of these examples are functionally equivalent
 *
 */

define("BR", '<br />');

// Define the array then print it out using the function
                    print_r()
// Use BR to separate each line

$array_1 = array(
    "0" => "FreeBSD",
    "1" => "OpenBSD",
    "2" => "NetBSD",
);

print_r($array_1);
echo BR;

$array_2 = [
    "0" => "FreeBSD",
    "1" => "OpenBSD",
    "2" => "NetBSD",
];

print_r($array_2);
echo BR;

// Arrays can use mixed key values - they do not have to start at 0

$array_3[5] = "FreeBSD";
$array_3["This is key 6"] = "OpenBSD";
$array_3[7] = "NetBSD";

print_r($array_3);
echo BR;

// Let PHP assign the key values

$array_4[] = "FreeBSD";
$array_4[] = "OpenBSD";
$array_4[] = "NetBSD";

print_r($array_4);
echo BR;
```

you do not recognize a function call in the later CMS sample code the script will be using a built in PHP function. The same apples to the javascript sample). See Listing 9.

Control structures

Control structures, along with Comparison / Logical operators provide the logic for our program. Example 3 is a good example of the if/else control structure.

These are only a very small subset and the most common of the extensive features available with PHP. To see the full list, please visit the PHP language guide at *http://www.php.net/manual/en/langref.php*.

Part 2. CMS Structure

See Table 2 – CMS directory structure. Create the directories and the 14 files as per the instructions for the example code under Part 1 – PHP fundamentals. Before we can start coding in earnest, we need to populate our MySQL database.

Create the following files, and create the database, table and our first page stored in the database: see Listings 10-12. Note that the Ipsum Lorem test should be on one line with no carriage returns or line feeds. Your editor may wrap this very long line. Create the the database, table and page as follows in Listing 13.

Table 1. *PHP Operators*

Example	Name	Result	Operator
-$a	Negation	Opposite of $a.	Arithmetic
$a + $b	Addition	Sum of $a and $b	
$a - $b	Subtraction	Difference of $a and $b	
$a * $b	Multiplication	Product of $a and $b	
$a / $b	Division	Quotient of $a and $b	
$a % $b	Modulus	Remainder of $a / $b	
$a = 3	Assignment	Sets $a to 3	Assignment
$a += 5	Assignment	Sets $a to 8	
$a = 'Hello '	Assignment	Sets $a to 'Hello'	
$a .= 'world'	Assignment	Sets $a to 'Hello world'	
$a == $b	Equal	TRUE if $a is equal to $b after type juggling.	Comparison
$a === $b	Identical	TRUE if $a is equal to $b, and they are of the same type.	
$a != $b	Not equal	TRUE if $a is not equal to $b after type juggling.	
$a <> $b	Not equal	TRUE if $a is not equal to $b after type juggling.	
$a !== $b	Not identical	TRUE if $a is not equal to $b, or they are not of the same type.	
$a < $b	Less than	TRUE if $a is strictly less than $b.	
$a > $b	Greater than	TRUE if $a is strictly greater than $b.	
$a <= $b	Less than or equal to	TRUE if $a is less than or equal to $b.	
$a <= $b	Greater than or equal to	TRUE if $a is greater than or equal to $b.	
$a and $b	And	TRUE if both $a and $b are TRUE.	Logical
$a or $b	Or	TRUE if either $a or $b is TRUE.	
$a xor $b	Xor	TRUE if either $a or $b is TRUE, but not both	
! $a	Not	TRUE if $a is not TRUE.	
$a && $b	And	TRUE if both $a and $b are TRUE.	
$a || $b	Or	TRUE if both $a and $b are TRUE.	
++$a	Pre-increment	Increments $a by one, then returns $a.	Inc / Dec
$a++	Post-increment	Returns $a, then increments $a by one.	
--$a	Pre-decrement	Decrements $a by one, then returns $a.	
$a--	Post-decrement	Returns $a, then decrements $a by one.	

Listing 8. *example8.php*

```
<?php

/*
 * example8.php
 * Demonstration of operator precedence
 *
 */

define("BR", '<br />');

$a = 1 + 5 * 3;
$b = (1 + 5) * 3;

echo '$a will evaluate to 16: ' . $a . BR;
echo '$b will evaluate to 18: ' . $b . BR;
```

Listing 9. *example9.php*

```
<?php

/*
 * example9.php
 * Demonstration of a function call
 *
 */

// As BR is a constant, this is available to our function directly

define("BR", '<br />');

// $pi is not available to our function, we will need to
                    access it by
// other methods

$pi = 22 / 7;

echo "Circumference with a diameter 5: " . print_circ1(5);
echo "Circumference with a diameter 10: " . print_circ2(10,$pi);

function print_circ1($diameter) {

    // print_circ1() will display $pi * $diameter
    // Define $pi as a global variable

    global $pi;

    // As BR is a global constant we can access it directly
    // Return our result to the main body of the program

    return $pi * $diameter . BR;
}

function print_circ2($diameter, $pi) {

    // print_circ2() will display $pi * $diameter
    // As BR is a global constant and $pi has been passed to our
    // function we can access them directly.
    // Return our result to the main body of the program

    return $pi * $diameter . BR;
}
```

Listing 10. *createdb.sql*

```
create database freebsdcms;
grant usage on *.* to bsduser@localhost identified by
                    'cmsdbpassword';
grant all privileges on freebsdcms.* to bsduser@localhost;
```

Listing 11. *createpagetbl.sql*

```
 CREATE TABLE if not exists pages (
        id INT NOT NULL AUTO_INCREMENT,
       PRIMARY KEY(id),
        title VARCHAR(50) NOT NULL,
        h1 VARCHAR(50),
       body TEXT
        );
```

Listing 12. *createpage.sql*

```
USE freebsdcms;
INSERT INTO pages
VALUES    (
    '',
    'My first page',
    'Page header',
    'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris
      interdum auctor tellus sed dignissim. Phasellus non orci massa,
      nec feugiat sem. Vestibulum molestie interdum
      bibendum. Nunc quis elit nulla, sit amet rutrum lorem.
                    Quisque
      odio est, sagittis nec accumsan ut, placerat sit
      amet lectus. Curabitur aliquam dignissim felis, a malesuada leo
      fringilla at. Sed ornare aliquet lacus, quis
      imperdiet augue mattis eu. Nulla porta odio ut erat
                    consectetur at
      molestie justo suscipit. Aenean convallis
      pellentesque nisl, vitae posuere mauris facilisis vitae.
                    Morbi in
      tellus nisl, vel facilisis diam.'
    );
```

Part 3. The Code

The following PHP files contain the code for out website. Create each file as per Table 2. See Listing 14.

global.css holds the style information for our site. Experiment with the font sizes, line spacing etc. to style the site to your liking. If you use Firefox, install the Firebug plugin to dynamically change the values, but if you want to make them pernanet you will need to edit this file. Also try renaming global.css and refreshing your browservs cache to see the effect of the styling on the site. See Listing 19.

The include files build our basic HTML header and footers, add the CSS add the CSS via global.css and load the Javascript. See Listing 20-22. The javascript files

Figure 3. *Our first page – index.php*

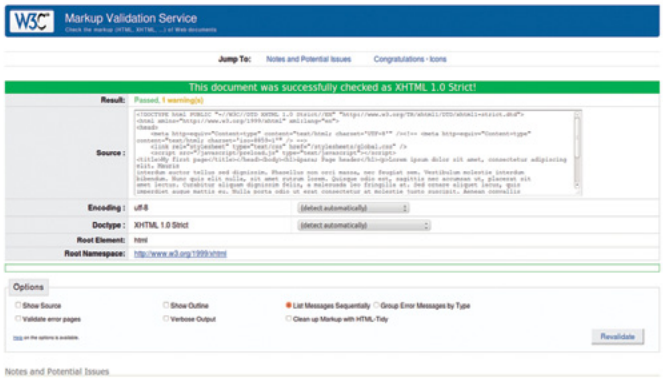


Figure 4. *The page validates*

Listing 13. *Creating the database, table and pages in MySQL*

```
#dev mysql -u root password 'cms-password' < createdb.sql
#dev mysql -u root password 'cms-password' < createpagetbl.sql
#dev mysql -u root password 'cms-password' < createpage.sql
```

Table 2. *CMS directory structure*

CMS Directory structure
All directories are under /usr/local/www/apache22/data

Directory / file	Purpose	Files
examples	Example PHP code	example1.php example2.php example3.php example4.php example5.php example6.php example7.php example8.php example9.php
includes	PHP includes for CMS. Each file contains specific functionality as named	cms.inc core.inc html.inc mysql.inc
index.php	Start page for our CMS	index.php
javascript	Javascript support for our website	postload.js preload.js
sql	Contains the SQL loader scripts for our website.	createdb.sql createpage.sql createpagetbl.sql
stylesheets	Holds the CSS stylesheets for the website	global.css
templates	Holds the templates for the website	header.inc footer.inc template.inc

Listing 14. *index.php*

```
<?php

/*
 * index.php
 * Index page for FreeBSD CMS
 *
 */

// Get required files

// Our global settings - Note need full path
require_once 'includes/cms.inc';
// Core functions
require_once INCLUDES.'core.inc';

// HTML functions
require_once INCLUDES.'html.inc';

// MySQL functions
require_once INCLUDES.'mysql.inc';

// Turn full debug functionality on if enabled

if(DEBUG){

    // Turn on full PHP error reporting
    error_reporting(E_ALL);

}else{

    // Hide all error messages
    error_reporting(0);

}

// Build page - use first record in database

$page['id'] = 1;

buildpage($page);
```

Listing 15. *cms.inc*

```
<?php

/*
 *
 * cms.inc
 * Contains default settings for our CMS
 *
 * NOTE: ¶ denotes a line wrapped - all code should be
                    on one line
 *
 */

// Set our timezone

date_default_timezone_set('Europe/London');

// Copyright details

define("LICENCE", 'licence.txt');
define("COPYRIGHT", 'Copyright &copy; 2013 Rob Somerville
                    ¶ me@merville.co.uk');
define("COPYYEAR", date('Y'));
define("COPYAUTH", 'Rob Somerville');
define("COPYEMAIL", 'me@merville.co.uk');

// Version

define("VERSION", 'Version 1.0 not for production use');

// Mode - If DEBUG is set to true, show errors and debug
                    info

define("DEBUG", TRUE);

// Where to find our files

define("TEMPLATES", 'templates/');
define("INCLUDES", 'includes/');
define("SQL", 'sql/');

// HTML tags that are orphaned and not defined in out
                    template files

define("BODY", '<body>');
define("HEAD", '</head>');

// MySQL details

define("DBSERVER", 'localhost');
define("DBUSER", 'bsduser');
define("DBPASSWORD", 'cmsdbpassword');
define("CMSDB", 'freebsdcms');
```

Listing 16. *core.inc*

```
<?php

/*
 *
 * core.inc
 * Contains core functions for our CMS
 *
 */

function buildpage($page) {

    // Builds a standard page

    $id = $page['id'];

    // Build the SQL and get the result

    $sql = "SELECT * FROM pages WHERE id='$id' LIMIT 1";
    $result = mysql_select($sql);

    // Output our page header

    outfile(TEMPLATES . 'header.inc');

    // Create our body

    $markup = '';
    $markup .= wraptag('title', $result[4]);
    $markup .= HEAD;
    $markup .= BODY;

    // If we are in debug mode, show an alert

    if(DEBUG){

        $debug = '&para; ';

    }else{

        $debug = '';
    }

    // Add to markup

    $markup .= wraptag('h1',$debug . $result[3]);
    $markup .= wraptag('p',$result[5]);
    $markup .= divclass(ahref(COPYRIGHT, LICENCE,
                'Copyright and ¶
                licence details'),'licence');

    // Output all our markup

    echo  $markup;

    // Output our HTML page footer

    outfile(TEMPLATES . 'footer.inc');

}

function outfile($file) {

    // Outputs template file to browser e.g header,
                footer, license etc.

    $fh = fopen($file, 'r');

    while (!feof($fh)) {
        echo fgets($fh);
    }

    fclose($fh);
}
```

Listing 17a. *html.inc*

```
<?php

/*
 *
 * html.inc
 * Contains core html functions for our CMS
 *
 */

function wraptag($tag, $text) {

    // Wraps $text with compliant tags
    // wraptag('p',sometext)
    // <p>sometext</p>

    return '<' . $tag . '>' . $text . '</' . $tag . '>';
}

function divclass($divcontent, $class, $id = '') {

    // Generates a div tag $text with compliant tags
    // divclass('content','class')
```

Listing 17b. *html.inc*

```
    // <div class="class">content</div>
    // divclass('content','class','id')
    // <div class="licence" id="id">content</div>

    if ($id != '') {

        $id = 'id="' . $id . '"';
    }

    return '<div class="' . $class . '" ' . $id . '>' .
    $divcontent . '</div>';
}

function ahref($text, $url, $title = '') {

    // Generates an href tag $text with compliant tags
    // ahref('Click here',freebsd.org)
    // <a href="http://freebsd.org" title="Click
                here">Click here</a>
    // ahref('Click here',freebsd.org,'Link title')
    // <a href="http://freebsd.org" title="Link
                title">Click here</a>

    if ($title == '') {
        $title = $text;
    }

    $ahref = '<a href="' . $url . '" title="' . $title .
                '">' . ¶
    $text . '</a>';

    return $ahref;
}
```

Listing 18. *mysql.inc*

```
<?php

/*
 *
 * mysql.inc
 * Contains MySQL functions for our CMS
 *
 */

function mysql_select($sql) {

$db = new mysqli(DBSERVER, DBUSER, DBPASSWORD, CMSDB);

if($db->connect_errno > 0){
    die('Unable to connect to database [' .
                $db->connect_error . ']');
}

if(!$result = $db->query($sql)){
    if(DEBUG){
        die('There was an error running the query [' .
                $db->error .
        ']');
    }else{
        die('');
    }
}

// Pass our results to an array to be returned

$r = array();

$r[] = $result->num_rows;      // No of rows returned
$r[] = $db->affected_rows;     // No of rows affected
                                  e.g. ¶
                            update /delete

while($row = $result->fetch_assoc()){
    $r[] = $row['id'];
    $r[] = $row['h1'];
    $r[] = $row['title'];
    $r[] = $row['body'];
}

// Free the result

$result->free();

return $r;

}
```

Listing 19. *global.css*

```
/* global.css - the site global stylesheet */

h1 {
    background-color: teal;
    float: left;
    color: white;
    padding: 21px;
    text-transform: uppercase;
}
p {
    float: left;
}
body {
    line-height: 160%;
    text-align: justify;
    float: left;
}
html {
    background: none repeat scroll 0 0 #F9F8F2;
    border: 1px solid;
    color: teal;
    font-family: Verdana;
    margin: 10px;
    padding: 20px;
}
.jstime, .licence {
    background: none repeat scroll 0 0 #EDEAC6;
    border: 1px solid #DADADA;
    color: slategrey;
    float: right;
    font-family: Verdana;
    font-size: x-small;
    margin-bottom: 5px;
    margin-right: 10px;
    margin-top: 10px;
    padding: 3px 10px;
}
```

Listing 20. *header.inc*

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//
EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
    <meta http-equiv="Content-type" content="text/html;
    charset='iso- 8859-1'" />
    <link rel="stylesheet" type="text/css" ¶
    href="/stylesheets/global.css" />
    <script src="/javascript/preload.js" ¶
    type="text/javascript"></script>
```

Listing 21. *footer.inc*

```
<script src="/javascript/postload.js" type="text/ ¶
javascript"></script></body></html>
```

Listing 22. *template.inc*

```
This can be empty, but needs to be created.

<!-- Template file -->
```

Listing 23. *preload.js*

```
/*
   preload.js
   Provides Javascript support
 */

// Call the function displaydate()

displaydate()


function displaydate(){

   // Displays the date and time in AM/PM format.

   var currentTime = new Date()
   var hours = currentTime.getHours()
   var minutes = currentTime.getMinutes()
   var month = currentTime.getMonth() + 1
   var day = currentTime.getDate()
   var year = currentTime.getFullYear()

   if (minutes < 10){
       minutes = "0" + minutes
   }

   var ampm = "";

   if(hours > 11){
       ampm = "PM"
   } else {
       ampm = "AM"
   }

   document.write("<div class=\"jstime\">" + day + "/" +
                  month + "/" +
   year + " " + hours + ":" + minutes + ampm + "</div>")
```

Listing 24. *postload.js*

```
/*
   postload.js
   Just an empty file with comments
 */
```

Useful links

- W3C Validator (by file upload) – *http://validator.w3.org*
- PHP documentation – *http://www.php.net/manual/en*
- W3 Schools – *http://www.w3schools.com*

are split into 2. Preload.js provides the date and time on each page, postload.js is just an empty file which provides hooks we will use later on in the series. See Listing 23-24.

Part 4. Our Simple CMS

Once you have entered the code as per table 2, point your browser at *http://yourserveripaddress/index.php*. You should see a page similar to Figure 3. Turn debug off and on in cms.inc, and the paragraph mark should disappear. If you copy the HTML source from the page (In you browser view source, select all, copy and paste into the W3C validator) the page should validate.

So what is our code doing?

The unformatted text is stored as plain text in our database table. Index.php forms the first page of our website, and loads our settings and functions from the include files. The first stage is to load our header from a plain text file, which is the HTML at the start of our page. The header file in turn loads the CSS and javascript, and returns control to index.php. We then query the database, wrap the text in the relevant HTML tags and output to our browser. We then close the HTML with our footer HTML. In the next part of our series we will develop the CMS further, passing parameters via our browser to load different pages, We will also start using our template file so that we can design our site the way we want it with separate blocks and regions.

ROB SOMERVILLE

Rob Somerville has been passionate about technology since his early teens. A keen advocate of open systems since the mid eighties, he has worked in many corporate sectors including finance, automotive, airlines, government and media in a variety of roles from technical support, system administrator, developer, systems integrator and IT manager. He has moved on from CP/M and nixie tubes but keeps a soldering iron handy just in case.

Hardening FreeBSD

with TrustedBSD and Mandatory Access Controls (MAC) Part 5

Most system administrators understand the need to lock down permissions for files and applications. In addition to these configuration options on FreeBSD, there are features provided by TrustedBSD that add additional layers of specific security controls to fine tune the operating system for multilevel security.

What you will learn…

- Configuration of the mac_ifoff, mac_portacl, and MAC LOMAC modules.

What you should know…

- Basic FreeBSD knowledge to navigate the command line
- Familiarity with loader.conf to enable kernel modules at boot

Since version 5.0 of FreeBSD, the TrustedBSD extensions have been included with the default install of the operating system. By default, this functionality is disabled and requires support to be compiled in or kernel modules to be loaded at boot time. For the purpose of this article, support will be loaded in with kernel modules already available with FreeBSD 9.1. Part 5 of the TrustedBSD series will cover the basic configuration of the `mac_ifoff`, `mac_portacl`, and `MAC LOMAC` modules.

Warning

Incorrect MAC settings can cause even the root user to not be able to login to the system. Be sure to run these tests on a VM or test machine to avoid any issues with production systems. This article assumes that a fresh install of FreeBSD 9.1 has been performed before continuing.

As in the previous articles, a certain set of users will help to illustrate how to use mandatory access controls (MAC). For the `mac_ifoff` module, the focus will be on network interfaces for the operating system. Listing 1 shows a basic setup for the required users for this article.

The `mac_ifoff` module provides an interesting feature that can turn off the network interfaces based on a sysctl value. To enable the `mac_ifoff` module add the following to `/boot/loader.conf` as detailed in Listing 2 to load the module at startup.

Once the system reboots, you will see "Security policy loaded: TrustedBSD MAC/ifoff (mac_ifoff)" in the output of dmesg. Manipulating the sysctl values for "mac_ifoff" allows for the network interfaces to be enabled or disabled. In order to make it easier to demonstrate this module, `tmux` will be installed to move between different user screens. The basic usage of `tmux` will be covered with additional information provided in the reference sec-

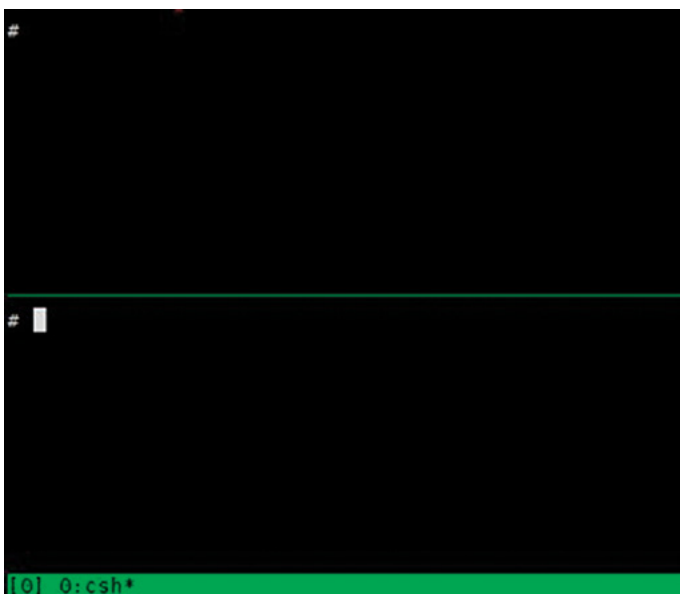


Figure 1. *Using tmux and multiple window panes*

tion. Listing 3 shows the installation steps for `tmux` using FreeBSD ports. Once `tmux` is installed, type `tmux` to invoke the terminal multiplexer and type the following:

```
Ctrl-b "
Ctrl-b "
```

This will open two window panes in one terminal, as seen in Figure 1.

Using Ctrl-b and the up and down arrow keys allows for the movement between window panes. Move to the top window, and run `id` then move to the middle pane and run `ssh user1@localhost`. Once logged in as `user1` type Ctrl-b and the arrow keys to select the root terminal and type what is shown in Listing 4.

When writing this article, ssh was used to remotely connect into the virtual machine with `tmux`. Figure 2 shows the result of both the root and `user1 ssh` connections being terminated immediately.

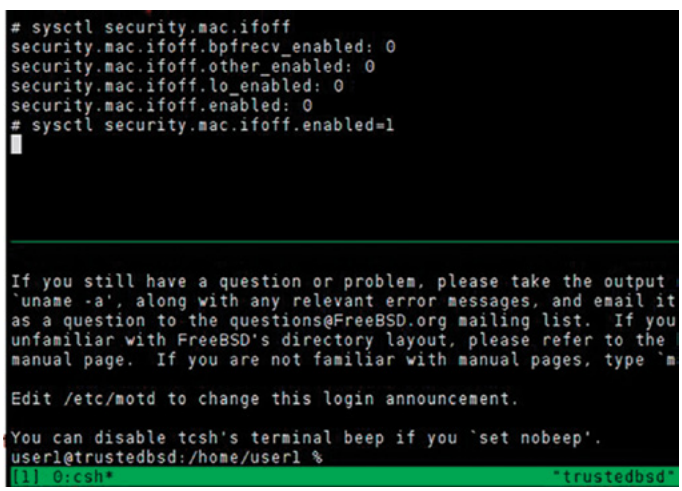


Figure 2. *mac_ifoff being enabled killed all network connections*

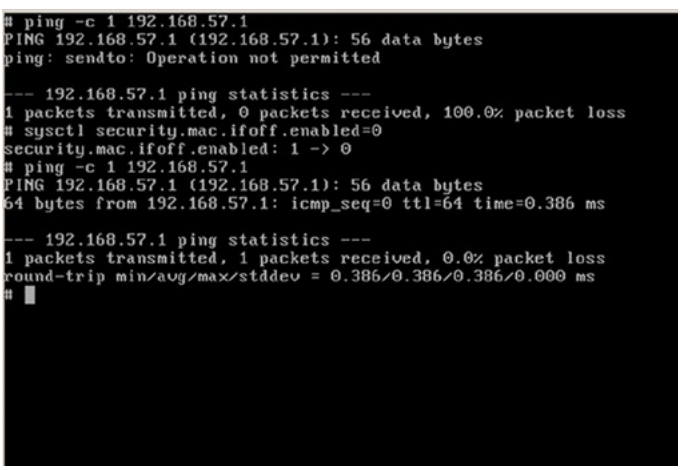


Figure 3. *mac_ifoff being disabled which allows network traffic*

Listing 1. *Users setup on FreeBSD*

```
# pw user add -n user1 -s /bin/csh -m
# pw user add -n user2 -s /bin/csh -m
# pw group add user-reg -M user1,user2
# passwd user1
Changing local password for user1
New Password:
Retype New Password:
# passwd user2
Changing local password for user2
New Password:
Retype New Password:
# groups user1
user1 user-reg
# groups user2
user2 user-reg
#
```

Listing 2. *Loading the mac_ifoff module on system startup (mac_portacl is added here as well)...*

```
# echo 'mac_ifoff_load="YES"' >> /boot/loader.conf
# echo 'mac_portacl_load="YES"' >> /boot/loader.conf
# echo 'security.mac.ifoff.lo_enabled=0' >> /etc/sysctl.
                    conf
# echo 'security.mac.ifoff.enabled=0' >> /etc/sysctl.
                    conf
# echo 'security.mac.ifoff.other_enabled=0' >> /etc/
                    sysctl.conf
# reboot
```

Listing 3. *Installing tmux*

```
# cd /usr/ports/sysutils/tmux/
# make -DBATCH install clean
===>  License BSD accepted by the user
===>  Found saved configuration for tmux-1.7_1
… output …
===>  Cleaning for tmux-1.7_1
#
```

Listing 4. *Enabled the mac.ifoff module*

```
# sysctl security.mac.ifoff.enabled=1
```

Listing 5. *Turning on the mac_portacl module which allows user1 to bind to TCP port 80*

```
(In the root window, type the following)
# sysctl security.mac.portacl.port_high=1023
security.mac.portacl.port_high: 1023 -> 1023
# sysctl net.inet.ip.portrange.reservedhigh=0
net.inet.ip.portrange.reservedhigh: 1023 -> 0
# id user1
uid=1001(user1) gid=1001(user1) groups=1001(user1)
# sysctl security.mac.portacl.rules=uid:1001:tcp:80
security.mac.portacl.rules:  -> uid:1001:tcp:80
#

(Use Ctrl-b and the arrow keys to move to the user1
                    window and type the following)
% nc -vnl 80
(root can connect to this port locally, with "nc -vn
                    127.0.0.1 80")
```

Listing 6. *Enable labels for the root file system (Warning: these commands are based on a default install with a single root partition and swap. If there are multiple partitions, edit the /etc/fstab by hand to ensure the root partition is set to ro)*

```
# sed -i '' -e 's/rw/ro/' /etc/fstab
# reboot
(Type 6 when it reboots to go into Single User Mode)
# tunefs -l enable /
# reboot
(Let the OS boot normally)
# mount -urw /
# sed -i '' -e 's/ro/rw/' /etc/fstab
(If there are multiple partitions, set root back to
                    readwrite 'rw')
# reboot
```

Listing 7. *Setup of files for mac_lomac*

```
(create some files)
# mkdir -p /data
# echo "HIGH" > /data/high.txt
# echo "LOW" > /data/low.txt

(change files low.txt and high.txt to lomac/high[low])
# setfmac -R lomac/equal /data
# setfmac lomac/high /data/high.txt
# setfmac lomac/low /data/low.txt
# setfmac lomac/high /usr/bin/vi
# vi /data/low.txt
ex/vi: Error: Log file: Permission denied
#
```

The functionality as highlighted in the FreeBSD handbook could be combined with an integrity checker such as `aide` or some other script. For example, monitoring `/etc/shadow` for changes to passwords and disabling remote connections when discovering an unauthorized change is a variation of checking the integrity of the system. From the console on the VM, Figure 3 shows how the interfaces are disabled until the sysctl value is changed to 0.

The `mac_portacl` module has interesting use cases for when administrators wish to allow under privileged users access to bind to privileged ports. This is beneficial for applications that do not provide a mechanism to drop root privileged. The `mac_portacl` module was added to loader.conf in an earlier procedure and should be loaded. Refer to the previous steps for launching tmux and using Ctrl- b" to create a window for root and `user1`. Run nc -vnl 80 as user1 and verify the output is the same as shown in Figure 4.

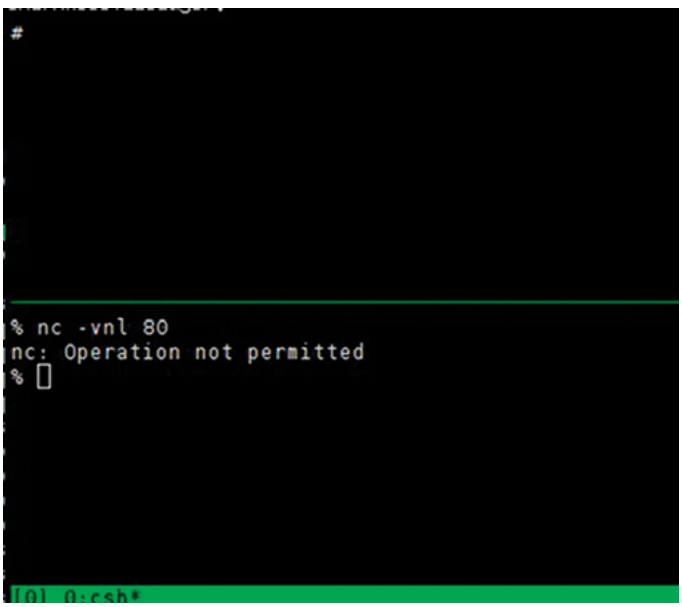


Figure 4. *user1 trying to bind to port 80*

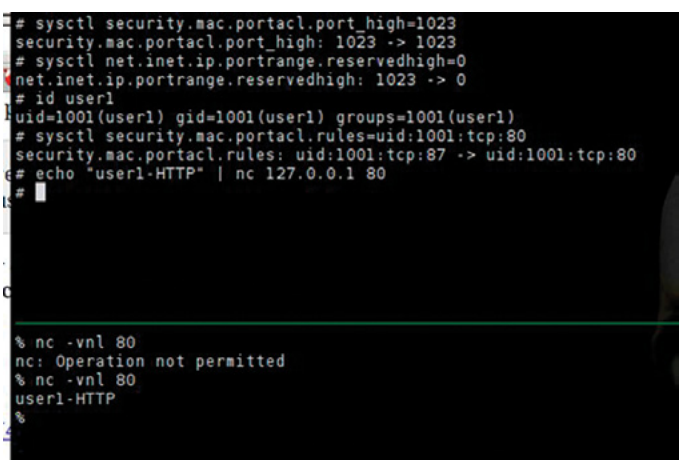


Figure 5. *Output from allowing user1 to bind to port 80*

nc stands for Netcat, one of the most versatile network tools. Using the features of the `mac_portacl` module, `user1` will be able to bind to the privileged port. Listing 5 contains the sysctl values that enable the module.

Figure 5 shows the output after allowing user1 to bind to port 80.

The last of the mandatory access controls builds upon the previously discussed `mac_biba` and is called `mac_lomac` module. This module extends the `mac_biba` module. Using previous steps from a previous article, we setup the proper labeling on the root filesystem and loading of the `mac_lomac` module at boot-up. Listing 6 contains the necessary steps for this process.

Listing 7 contains some commands to test the mac_lomac settings using vi.

In this example, vi is set to a `lomac/high` profile, which does not allow it to continue opening the file once it accesses a `lomac/low` file. All of the information presented in these articles is a start to providing examples on how to implement Mandatory Access Controls with TrustedBSD on FreeBSD. Even as the FreeBSD code base moves into version 10, these features are still labeled as experimental and are to be thoroughly tested before deploying in production. Future security enhancements may incorporate these modules or replace them with newer code to better fine tune access controls on FreeBSD.

References

- FreeBSD Handbook – Mandatory Access Control: *http://www.freebsd.org/doc/handbook/mac.html*
- MAC LOMAC Module: *http://www.freebsd.org/doc/handbook/mac-lomac.html*
- MAC mac-portacl Module: *http://www.freebsd.org/doc/handbook/mac-portacl.html*
- MAC ifoff Module: *http://www.freebsd.org/doc/handbook/mac-ifoff.html*
- Tmux: *http://tmux.sourceforge.net/*
- TrustedBSD: *http://www.trustedbsd.org/*

MICHAEL SHIRK

Michael Shirk is a BSD zealot who has worked with OpenBSD and FreeBSD for over 7 years. He works in the security community and supports Open Source security products that run on BSD operating systems. Michael is the Chief Executive Manager of Daemon Security Inc., a company which provides security solutions utilizing the BSD operating systems: http://www.daemon-security.com.

BSD MAGAZINE

In the next issue:

ALL ABOUT FREENAS!

- Article by John Hixson, author of the FreeNAS plugins
- Dru Lavigne article on plugins and encryption in FreeNAS 8.x
- Interview with Alfred Perlstein, Vice President, Software Engineering
- Article by Mark VonFange, Professional Services Manager at iXsystems

Next issue is coming in April!

Need help with compliance?

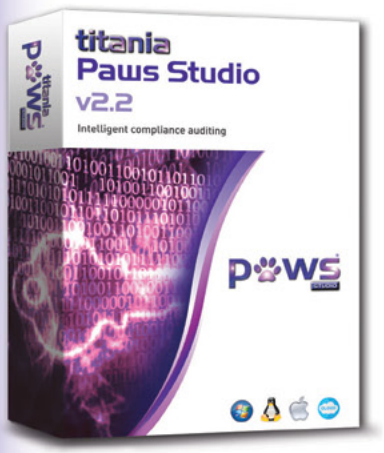
Your custom policy

PCI NERC NORTH AMERICAN ELECTRIC RELIABILITY CORPORATION SANS

Use Paws Studio

to audit your workstations and servers

Paws Studio is efficient, easy to use and cost effective. The software provides comprehensive reporting and management summaries to appeal to all levels of your organization.



...from the creators of award winning Nipper Studio software

With Paws Studio you can:

1. Produce remote compliance audits using remote connectivity or audit offline with our unique Data Collector
2. Use the Remedy Table to quickly solve potential compliance issues
3. Create and modify your own policies using the Paws definition editor

Compliance Checklist	Paws Studio
Antivirus	✓
Spyware	✓
Audit Policy	✓
Files & Directories	✓
Windows Firewalls	✓
Password Policies	✓
Password Warnings	✓
Permissions	✓
Registry Settings	✓
Software Updates	✓
Installed Software	✓
Illegal Software	✓
Software Versions	✓
User Policies	✓
User Rights	✓

titania PAWS STUDIO

CLOUD

enquiries@titania.com
T: +44 (0) 1905 888785

Headquarters:
San Jose, CA

855.GREP.4.IX | Contact Us

99% Compatibility

online now...

IXSYSTEMS AND YOU ARE THE PERFECT MATCH

SHARED INTERESTS

- [x] Enterprise Storage Solutions
- [x] Personalized Customer Service
- [x] Bold New Information Technology

I'm a: Storage Reseller

In: The EU

Looking for: Storage Solutions to Sell

- A Technology Partner
- More Technical Experience
- New Business Opportunities

Visit Today!

iXsystems

Technology Partner Seeking Resellers/Integrators for TrueNAS™ Storage Appliance

WWW.IXSYSTEMS.COM/PERFECTMATCH