MAGAZINE BSD

FOR NOVICE AND ADVANCED USERS

CARP on FreeBSD

HOW TO USE DEVD TO TAKE ACTION ON KERNEL EVENTS

FREEBSD PROGRAMMING PRIMER

UNIX BASICS FOR SECURITY PROFESSIONALS

OPENBSD 5.4 AS A TRANSPARENT HTTP/HTTPS PROXY USING PF AND RELAYD

UNIX KERNEL

GHOSTBSD: A USERFRIENDLY, LIGHTWEIGHT BSD ALTERNATIVE

HOW SECURE CAN SECURE SHELL (SSH) BE?

VOL.7 NO.12
ISSUE 12/2013(53)
1898-9144

iXsystems®

855-GREP-4-IX
www.iXsystems.com
Enterprise Servers and Storage for Open Source

✓ Rock-Solid Performance

✓ Professional In-House Support

E5-2600

High Performance, High Density Servers for Data Center, Virtualization, & HPC

MODEL: iXR-22X4IB

http://www.iXsystems.com/e5

768GB of RAM in 1U

KEY FEATURES

iXR-22X4IB

- Dual Intel® Xeon® Processors E5-2600 Family per node
- Intel® C600 series chipset
- Four server nodes in 2U of rack space
- Up to 256GB main memory per server node
- One Mellanox® ConnectX QDR 40Gbp/s Infiniband w/QSFP Connector per node
- 12 SAS/SATA drive bays, 3 per node
- Hardware RAID via LSI2108 controller
- Shared 1620W redundant high-efficiency Platinum level (91%+) power supplies

iXR-1204+10G

- Dual Intel® Xeon® Processors E5-2600 Family
- Intel® C600 series chipset
- Intel® X540 Dual-Port 10 Gigabit Ethernet Controllers
- Up to 16 Cores and 32 process threads
- Up to 768GB main memory
- Four SAS/SATA drive bays
- Onboard SATA RAID 0, 1, 5, and 10
- 700W high-efficiency redundant power supply with FC and PMBus (80%+ Gold Certified)

Call iXsystems toll free or visit our website today! **1-855-GREP-4-IX | www.iXsystems.com**

High-Density iXsystems Servers powered by the Intel® Xeon® Processor E5-2600 Family and Intel® C600 series chipset can pack up to 768GB of RAM into 1U of rack space or up to 8 processors - with up to 128 threads - in 2U.

On-board 10 Gigabit Ethernet and Infiniband for Greater Throughput in less Rack Space.

Servers from iXsystems based on the Intel® Xeon® Processor E5-2600 Family feature high-throughput connections on the motherboard, saving critical expansion space. The Intel® C600 Series chipset supports up to 384GB of RAM per processor, allowing performance in a single server to reach new heights. This ensures that you're not paying for more than you need to achieve the performance you want.

The iXR-1204 +10G features dual onboard 10GigE + dual onboard 1GigE network controllers, up to 768GB of RAM and dual Intel® Xeon® Processors E5-2600 Family, freeing up critical expansion card space for application-specific hardware. The uncompromised performance and flexibility of the iXR-1204 +10G makes it suitable for clustering, high-traffic webservers, virtualization, and cloud computing applications - anywhere you need the most resources available.

For even greater performance density, the iXR-22X4IB squeezes four server nodes into two units of rack space, each with dual Intel® Xeon® Processors E5-2600 Family, up to 256GB of RAM, and an on-board Mellanox® ConnectX QDR 40Gbp/s Infiniband w/QSFP Connector. The iXR-22X4IB is perfect for high-powered computing, virtualization, or business intelligence applications that require the computing power of the Intel® Xeon® Processor E5-2600 Family and the high throughput of Infiniband.

HIGH & Throughput INCREDIBLE Performance Density

IXR-1204+10G: **10GbE On-Board**

4 Server Nodes in 2U

IXR-22X4IB

iXsystems®

intel inside™ Xeon®

Call iXsystems toll free or visit our website today! **1-855-GREP-4-IX | www.iXsystems.com**

Dear BSD Readers,

I hope that this year was fruitful and prosperous for you and it is my hope that the next one will be even better. Before the year ends, I would like to thank all the great people who made the 2013 year beautiful for me, BSD Team and BSD magazine.

We, the BSD team, would like to wish you continuous prosperity, development and success as well as good health and happiness to you and your loved ones.

Let this Christmas be a good time to think about our past and future projects to make them even better in the forthcoming year.

Spread the happiness all around you!
Merry Christmas and A Happy New Year,
Ewa & BSD Team

MAGAZINE BSD

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

Contributing:
Michael Shirk, Andrey Vedikhin, Petr Topiarz, Charles Rapenne, Anton Borisov, Jeroen van Nieuwenhuizen, José B. Alós, Luke Marsden, Salih Khan, Arkadiusz Majewski, BEng

Top Betatesters & Proofreaders:
Annie Zhang, Denise Ebery, Eric Geissinger, Luca Ferrari, Imad Soltani, Olaoluwa Omokanwaye, Radjis Mahangoe, Mani Kanth, Ben Milman, Mark VonFange

Special Thanks:
Annie Zhang
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

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

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

Configuring a Highly Available Service on FreeBSD

– Part 2: CARP and devd

In the first part of this series, we learned how to make high availability (HA) storage on FreeBSD using HAST. We learned how to control HAST and how to recover from failures. However, all those actions were still manual actions. In this second part of the series, we will learn how two basic building blocks, CARP and devd, work and how we can use them in the final part of our series to automate the failover of our NFS server.

What you will learn…

- How to configure CARP on FreeBSD
- How to use devd to take action on kernel events

What you should know…

- How to login to FreeBSD
- How to edit files on FreeBSD
- Basic understanding of network configuration
- The nfs-01 and nfs-02 machines from part 1

CARP stands for common address redundancy protocol and makes it possible to share an IP (IPv4 and/or IPv6) address between multiple hosts in so called 'redundancy groups'. The IP that is shared between the hosts in the redundancy group resides on the master host for that group. In case the master goes down, the other members (backups) in the redundancy group will elect a new master. This master will then 'take' the shared IP.

That, of course, sounds nice, but how does that help us? Well, to implement our failover NFS service, we need an IP address for this service to reside on the host that will service the NFS requests. The host that will service the NFS request would be the primary HAST node. Also, in case of a HAST failover, we would like the service IP to switch to the new primary HAST node. So, if we are able to keep the CARP master state and the HAST primary state in sync with each other, we would always have the shared IP, which we can use for the NFS service, on the host that is the primary HAST node.

How to configure CARP

CARP can be configured by using the *ifconfig* command as described in listing 1. Note that in our example, setup nfs-01 will have the IP 192.168.254.1 and nfs-02 will have the IP 192.168.254.2. Both with a /24 netmask.

The first command for nfs-01 in Listing 1 creates a carp interface called *carp0* on that host. The second command configures this newly created *carp0* interface with the correct parameters. The first parameter *vhid* is the virtual host ID, which uniquely identifies the redundancy group on the network and therefore should be the same on all hosts in the same redundancy group. In our example, we use a *vhid* of 1. The second parameter pass is used to authenticate the carp advertisements and is in our case set to *bsdmag*. This parameter should also be the same on all hosts in the same redundancy groups. Although the *pass* parameter is optional, it is wise to set it, otherwise machines not part of the redundancy group can easily send out bogus carp traffic to disrupt our redundancy group. The third parameter is *advbase*, which specifies the base advertisement interval in seconds. These advertisements are needed to determine if the master is still up and if not to elect a new master. The fourth parameter *advskew* is closely related to the *advbase* parameter; when set, it adds a small amount of time to *advbase* so that advertisements are sent out a little less frequently than specified by *advbase*. This fourth parameter differs in our example for nfs-01 and nfs-02. It is higher for nfs-02 so that nfs-01 will become the master if both hosts come online at the same time, because nfs-01 will send out its advertisements more frequently. The last parameter specifies the shared IP to use and the network it resides on. In our case, the shared IP is 192.168.254.100 with a /24 netmask. This IP will become active on the master on the interface that is in the same network as specified for the carp0 interface. If, for example, nfs-01 is the master, the shared IP 192.168.254.100 will become available on the same interface as 192.168.254.1 as that interface is in the same network.

Listing 1. *configuring CARP on our hosts*

```
nfs-01# ifconfig create carp0
nfs-01# ifconfig carp0 vhid 1 pass bsdmag advbase 1
    advskew 10 192.168.254.100/24
nfs-01# ifconfig carp0
  carp0: flags=49<UP,RUNNING> metric 0 mtu 1500
         inet 192.168.254.1 netmask 0xffffff00
         carp: MASTER vhid 1 advbase 1 advskew 20
nfs-02# ifconfig create carp0
nfs-02# ifconfig carp0 vhid 1 pass bsdmag advbase 1
    advskew 20 192.168.254.100/24
nfs-02# ifconfig carp0
  carp0: flags=49<UP,RUNNING> metric 0 mtu 1500
         inet 192.168.254.2 netmask 0xffffff00
         carp: BACKUP vhid 1 advbase 1 advskew 10
```

The third command not only shows us the configuration of our *carp0* interface, but also shows whether the interface is in the master or in the backup state in the line starting with *carp:* Note that the password used is not visible. The configuration of the *carp0* interface on the nfs-02 is analog to the configuration of the *carp0* interface on the nfs-01 with the earlier mentioned difference of the advskew parameter.

Making CARP reboot proof

Now that we know how to configure CARP, we want to make sure that our configuration becomes reboot proof. This can be done by adding a few lines to `/etc/rc.conf`. In Listing 2 you can find the lines we would need to add to `/etc/rc.conf` on the nfs-01 server. The first line makes sure our *carp0* device will be created on boot. The second line configures the *carp0* interface and is identical to the parameters we passed to the ifconfig command for *carp0* earlier. It is left as an exercise for the reader to find the correct configuration for the nfs-02 server.

Listing 2. *Making the CARP configuration reboot proof on nfs-01*

```
cloned_interfaces="carp0"
ifconfig_carp0="vhid 1 pass bsdmag advbase 1 advskew 10
    192.168.254.100/24"
```

Testing CARP

After we have made our CARP configuration reboot proof, it is good to perform some basic tests to see whether the failover of the shared IP works as expected. First we will force a switch of the shared IP from our current master (nfs-01) to nfs-02. When that is complete and nfs-02 has indeed become the new master we will force the master back to the nfs-01. In addition to testing, the commands in listing 3 that describe these actions are also useful in the case when a manual switch has to be forced. Please be especially aware of the host you have to execute the commands on to trigger the failover. An important note to make is that in case you are building this setup on a virtual platform, broadcast traffic should be allowed for the virtual machines or CARP won't work. Allowing broadcast traffic is not the default setting for all virtualisation solutions.

What is DEVD?

Devd is the device state change daemon and it is a system daemon that runs in the background and hooks into the *devctl* device driver. When a change occurs in the device configuration tree, this device driver will pass this information to devd. Devd will parse this message and will look into its action list for an action to execute. This way devd provides a way to have userland programs run when certain kernel events happen. The default configuration file for devd is `/etc/devd.conf`. By default this file includes the options to also scan the `/etc/devd` and `/usr/local/etc/devd` directories for devd configuration files.

Listing 3. *Testing the CARP failover*

```
Moving the master from nfs-01 to nfs-02 (commands
    executed on nfs-01)
nfs-01# ifconfig carp0 down
nfs-01# ifconfig carp0 up

Checking the status on both hosts (commands executed
    on nfs-01 and nfs-02)
nfs-01# ifconfig carp0
nfs-02# ifconfig carp0

Moving the master from nfs-02 to nfs-01 (commands
    executed on nfs-02)
nfs-02# ifconfig carp0 down
nfs-02# ifconfig carp0 up

And again checking the status on both hosts (commands
    executed on nfs-01 and nfs-02)
nfs-01# ifconfig carp0
nfs-02# ifconfig carp0
```

Devd syntax

To explain the syntax of devd we will make a slight side step by looking at the devd configuration shown in Listing 4. What this configuration does is log a message to syslog when a USB device is attached. Let's inspect this configuration a little bit further. The first line *notify 0* indicates that an action should be taken when the kernel sends an event notification to the user land. The priority of this rule is 0. This priority is used to decide which action to take when more than one rule matches. If more than one rule matches the rule with the lowest number is executed. To restrict the cases in which our action will be executed we use the *match* clauses on line 2 till 4 to restrict it. Line 2 matches the event message against the system it is coming from, in this case the USB system. So all events that are not from the USB system will not trigger the action. The next line restricts the action to a subsystem of the USB system. In this case it is the *interface* subsystem, so the event should come from a USB interface to trigger our action. The last match rule of Listing 4 restricts the type of event, in this case the attachment of a device. Last but not least, we have line 5, which specifies the action to execute. In this case we log a message to syslog to notify us that a USB device has been attached, but an action line can call every command you like. More information about all the systems, subsystems, types and action you can handle with devd can be found in the *devd.conf* manual page.

Listing 4. *A devd configuration for USB events*

```
notify 0 {
      match "system"        "USB" ;
      match "subsystem"  "INTERFACE" ;
      match "type"            "ATTACH" ;
      action "logger USB device attached" ;
} ;
```

Listing 5. *Configuring devd for CARP*

```
notify 30 {
      match "system"        "IFNET" ;
      match "subsystem"  "carp*" ;
      match "type"            "LINK_UP" ;
      action "logger -t bsdmag $subsystem device is
   UP" ;
} ;

notify 30 {
      match "system"        "IFNET" ;
      match "subsystem"  "carp*" ;
      match "type"            "LINK_DOWN" ;
      action "logger -t bsdmag $subsystem device is
   DOWN" ;
} ;
```

Configuring devd for CARP

Now that we have a basic grasp of how to use devd to take actions on kernel events we can start to configure devd to handle events originating from our CARP interfaces. In listing 5 we see a configuration that will log to syslog when we receive a LINK_DOWN or a LINK_UP event from our carp0 interface. Because a CARP device is a network system, the system we have to use in our match rule is IFNET. Noteworthy is the wildcard match we use in the subsystem, hence the action will run for an event on any carp interface that matches the type. To separate between the link going up and the link going down we created 2 statements, one for the LINK_UP and one for the LINK_DOWN event. Also interesting is the action line we use. Again, we use *logger* to log a message to syslog, but we also use the *$subsystem* variable available to log the exact subsystem that the event came from, so in the log we will see which interface generated the event. By putting the configuration from Listing 5 in `/etc/devd/hast.conf` and by restarting devd with *service devd restart* we make sure it will be used by devd.

Testing our devd configuration

Testing of our devd configuration is more or less analog to the initial testing of our CARP configuration, so we can use the commands from listing 3 again. However, to see if our devd recipe for CARP worked, we should not only check the status of our *carp0* interface with *ifconfig carp0*, but also check `/var/log/messages` to see if the log messages we configured in listing 5 are indeed written to the syslog correctly, so we are sure devd is configured correctly. Take good note of when a CARP interface sends the LINK_UP type and when it sends the LINK_DOWN type of event. You will see that the CARP interface sends the LINK_UP message via devd only when it becomes the master and the LINK_DOWN message when it goes down and when it becomes the backup.

Conclusion

In this second part of the series we introduced CARP and devd. We learned how to configure CARP, and how to make an IP highly available with it. We also learned what devd is and how to take actions on kernel events by using devd. Especially, we learned how to run a script from devd in case of a CARP failover. Now that we know how to configure HAST, CARP and devd we can put all these building blocks together in the final part of our series in which we will create the highly available NFS server and the failover script to call from devd.

Questions received from readers

Q: How highly available is HAST when a filesystem check can take ages on a large filesystem?

During the filesystem check HAST is of course unavailable. But you really should do this check, to be sure your filesystem is in a consistent state. Otherwise you might run into problems later that cause much more downtime than the filesystem check would take. To reduce the time spent filesystem checking, it is also good practice to always use a journaling filesystem on your HAST devices. One important point to keep in mind is also that highly available does not mean *always* available, so yes, in case of a node failure you probably will have some downtime, but significantly less than when you would need to rebuild your machine and restore from backup. Also your dataloss will probably be significantly less.

Q: But what if I do need my filesystem to always be available?

In that case you should probably look into gluster (RedHat Linux), LustreFS or CEPH, which are clustered/distributed filesystems but all have a focus on Linux unfortunately.

JEROEN VAN NIEUWENHUIZEN

Jeroen van Nieuwenhuizen works as a unix consultant for Snow. Besides playing with FreeBSD, his free time activities include cycling, chess and ice skating.

FreeBSD Programming Primer – Part 11

In the penultimate part of our series on programming, we will look at using the Netbeans Integrated Development Environment to debug and edit our CMS.

What you will learn…

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

What you should know…

- BSD and general PC administration skills

Unfortunately, the Internet gremlins have got me at the moment so this how-to is going to be very short. My local telco is currently rolling out fibre in the area, and my ADSL internet connection is very unreliable, but hopefully I will be able to wrap up the programming primer in part 12 with a bumper edition.

While debugging at the command line using echo statements or commenting out code is possible, a more frequent scenario is that our project will be residing on a remote server and we will need to see the actual processes in action. Often developers will have a local copy of the LAMP stack on their PC or laptop, so that they can debug locally. However, what happens when our development environment is on a laptop and the code is on a remote server? A frequent approach is to use an Integrated Development Environment (IDE) with a built in file transfer utility. Coupled with Xdebug, which supports PHP, we can download our remote code and debug (step through) each line, examine variables etc. To do this, we will need to install Xdebug on our server and install the IDE of our choice on an available local PC. This can be FreeBSD, Windows or Linux, but in my case I was using an Ubuntu desktop. The IDE installation will vary from environment to environment, full details can be found at *https://netbeans.org*. The IP address of of my desktop PC for this exercise was 192.168.0.123.

Installing Xdebug

Rather than using the FreeBSD provided software, I downloaded the latest version from http://xdebug.org. The reason for this is that in the past I have had problems getting the standard packaged version of Xdebug working with certain distro's, where as the latest Xdebug

Listing 1. *Install Xdebug*

```
tar -xvzf xdebug-2.2.3.tgz
cd xdebug-2.2.3
phpize
./configure -enable-xdebug
make
cd modules

cp xdebug.so /usr/local/lib/php/20100525/
touch /var/log/xdebug.log
chmod 666 /var/log/xdebug.log
touch /user/local/etc/php/xdebug.ini
```

Listing 2. */user/local/etc/php/xdebug.ini*

```
zend_extension=/usr/local/lib/php/20100525/xdebug.so
xdebug.remote_enable=1
xdebug.remote_host="192.168.0.123"
xdebug.remote_port=9000
xdebug.remote_handler="dbgp"
xdebug.remote_mode=req
xdebug.profiler_enable = 1
xdebug.remote_log=/var/log/xdebug.log
```

Listing 3. *Restarting Apache*

```
/usr/local/etc/rc.d/apache22 stop
/usr/local/etc/rc.d/apache22 start
```

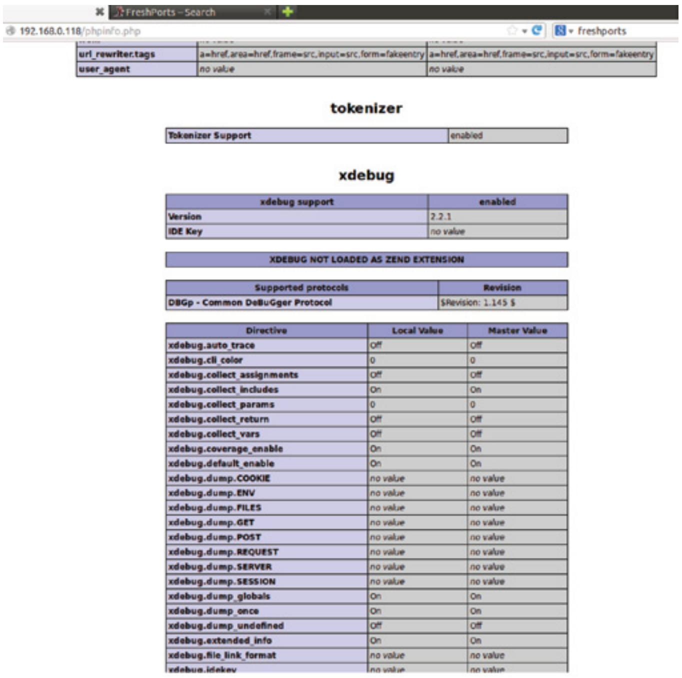
and latest Netbeans IDE always seem to work OK together. Once you have downloaded the latest version of the tarball (Currently xdebug-2.2.3.tgz) into your home directory, on the remote server (192.168.0.118) as root, perform the following (Listing 1).

Add the following to `/user/local/etc/php/xdebug.ini` (Listing 2).

Replace 192.168.0.123 with the IP address of your client machine.

Restart Apache (Listing 3).

If we now login as admin and visit our PHPinfo page at *http:/192.168.0.118/phpinfo.php*, we should see that Xdebug is installed and running (Figure 1). If you have not already done so, download and install Netbeans on a local PC of your choice. You will need a working Java installation and Firefox installed for this to work.



url_rewriter.tags	a=href,area=href,frame=src,input=src,form=fakeentry	a=href,area=href,frame=src,input=src,form=fakeentry
user_agent	no value	no value

tokenizer

Tokenizer Support	enabled

xdebug

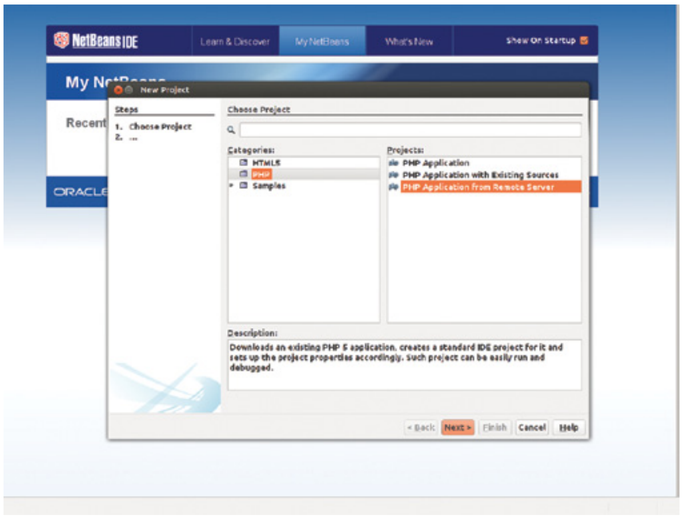
xdebug support	enabled
Version	2.2.1
IDE Key	no value

XDEBUG NOT LOADED AS ZEND EXTENSION

Supported protocols	Revision
DBGp - Common DeBuGger Protocol	$Revision: 1.145 $

Directive	Local Value	Master Value
xdebug.auto_trace	Off	Off
xdebug.cli_color	0	0
xdebug.collect_assignments	Off	Off
xdebug.collect_includes	On	On
xdebug.collect_params	0	0
xdebug.collect_return	Off	Off
xdebug.collect_vars	Off	Off
xdebug.coverage_enable	On	On
xdebug.default_enable	On	On
xdebug.dump.COOKIE	no value	no value
xdebug.dump.ENV	no value	no value
xdebug.dump.FILES	no value	no value
xdebug.dump.GET	no value	no value
xdebug.dump.POST	no value	no value
xdebug.dump.REQUEST	no value	no value
xdebug.dump.SERVER	no value	no value
xdebug.dump.SESSION	no value	no value
xdebug.dump_globals	On	On
xdebug.dump_once	On	On
xdebug.dump_undefined	Off	Off
xdebug.extended_info	On	On
xdebug.file_link_format	no value	no value
xdebug.idekey	no value	no value

Figure 1. *PHP Xdebug installed*



Steps
1. Choose Project
2. ...

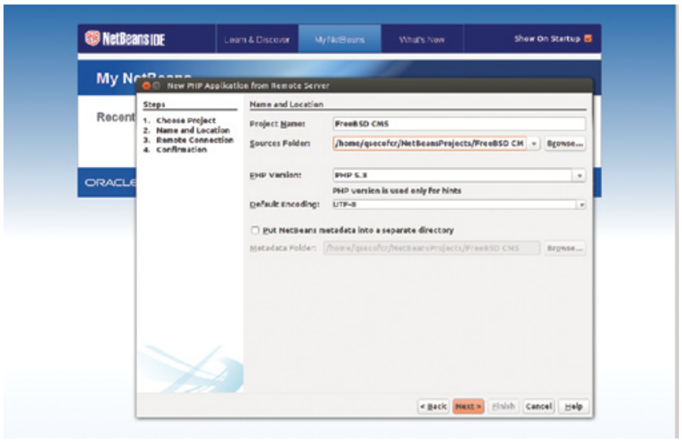
Choose Project

Categories: HTML5, PHP, Samples

Projects: PHP Application, PHP Application with Existing Sources, PHP Application from Remote Server

Description: Downloads an existing PHP 5 application, creates a standard IDE project for it and sets up the project properties accordingly. Such project can be easily run and debugged.

Figure 2. *Create a new project with PHP application on remote server*



Steps
1. Choose Project
2. Name and Location
3. Remote Connection
4. Confirmation

Name and Location

Project Name: FreeBSD CMS

Sources Folder: /home/qsecofcr/NetBeansProjects/FreeBSD CM

PHP Version: PHP 5.3

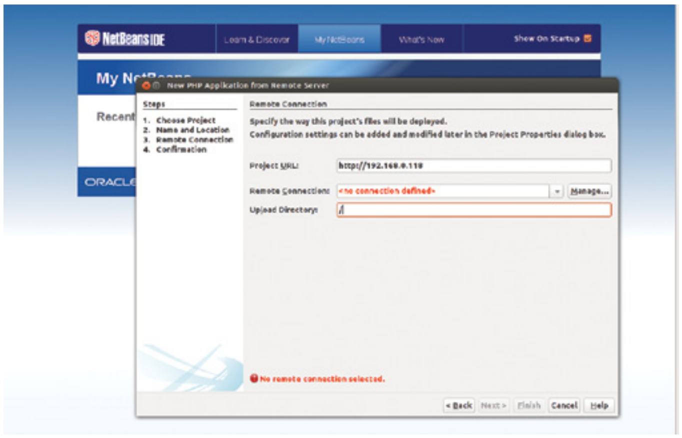
PHP version is used only for hints

Default Encoding: UTF-8

☐ Put NetBeans metadata into a separate directory

Metadata Folder: /home/qsecofcr/NetBeansProjects/FreeBSD CMS

Figure 3. *Give the project a name*



Steps
1. Choose Project
2. Name and Location
3. Remote Connection
4. Confirmation

Remote Connection

Specify the way this project's files will be deployed.
Configuration settings can be added and modified later in the Project Properties dialog box.

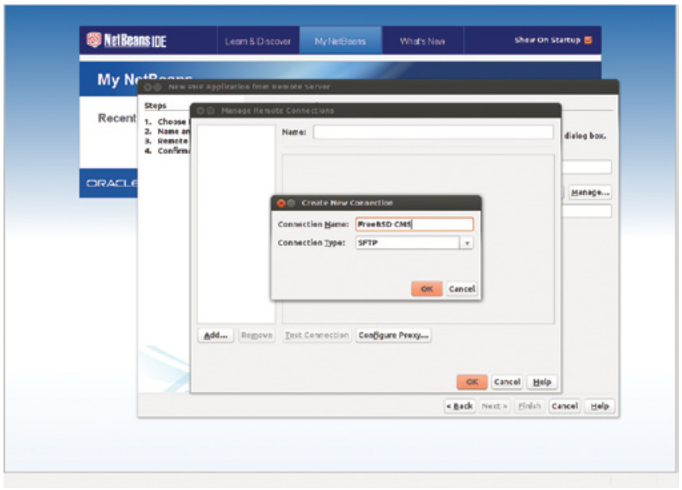
Project URL: http://192.168.0.118

Remote Connection: <no connection defined>

Upload Directory: /

No remote connection selected.

Figure 4. *Create a new remote connection by clicking on Manage*



Name:

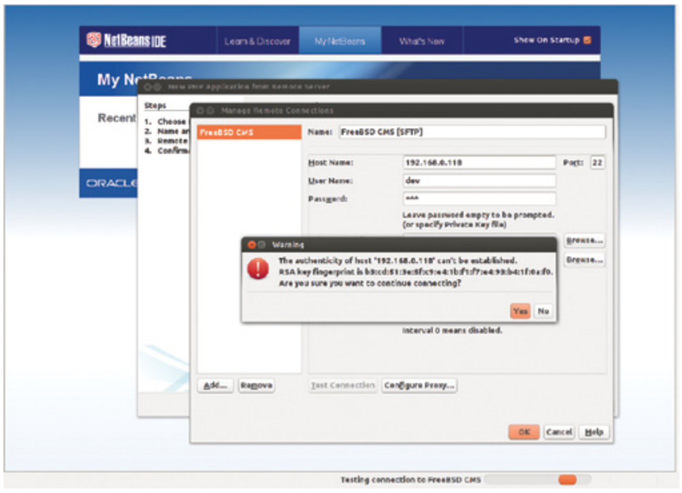
Connection Name: FreeBSD CMS

Connection Type: SFTP

Figure 5. *Create a new SFTP connection (SSH must be running on Port 22 of your server)*

Useful links

- Xdebug: *http://xdebug.org*
- Netbeans: *http://php.net/manual*



Manage Remote Connections

FreeBSD CMS

Name: FreeBSD CMS [SFTP]

Host Name: 192.168.0.118 Port: 22

User Name: dev

Password: ***

Leave password empty to be prompted.
(or specify Private Key file)

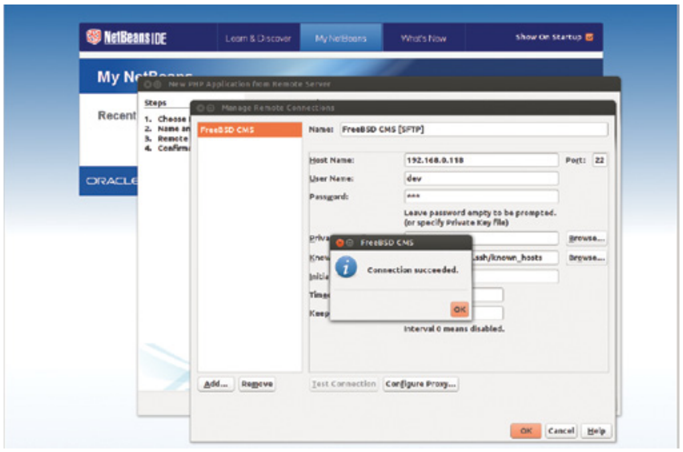
Warning

The authenticity of host "192.168.0.118" can't be established.
RSA key fingerprint is b3:cd:51:3e:8f:c9:e4:1b:f1:f7:e4:93:b4:1f:0a:f0.
Are you sure you want to continue connecting?

Interval 0 means disabled.

Testing connection to FreeBSD CMS

Figure 6. *Testing the connection*



Manage Remote Connections

FreeBSD CMS

Name: FreeBSD CMS [SFTP]

Host Name: 192.168.0.118 Port: 22

User Name: dev

Password: ***

Leave password empty to be prompted.
(or specify Private Key file)

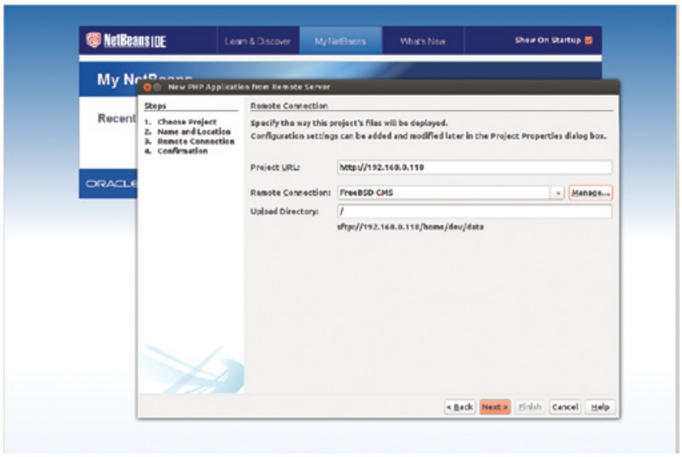
.ssh/known_hosts

FreeBSD CMS

Connection succeeded.

Interval 0 means disabled.

Figure 7. *A successful connection*



New PHP Application from Remote Server

Steps
1. Choose Project
2. Name and Location
3. Remote Connection
4. Confirmation

Remote Connection

Specify the way this project's files will be deployed.
Configuration settings can be added and modified later in the Project Properties dialog box.

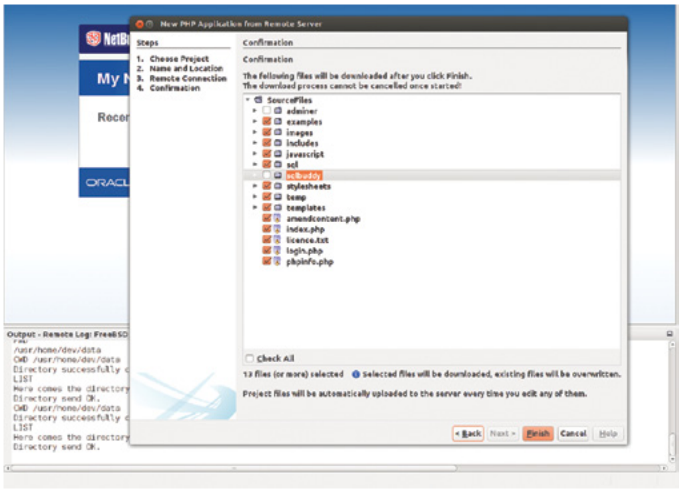
Project URL: http://192.168.0.118

Remote Connection: FreeBSD CMS

Upload Directory: /

sftp://192.168.0.118/home/dev/data

Figure 8. *The final settings of the remote project. Replace with your server IP address as required*



New PHP Application from Remote Server

Steps
1. Choose Project
2. Name and Location
3. Remote Connection
4. Confirmation

Confirmation

The following files will be downloaded after you click Finish.
The download process cannot be cancelled once started!

```
SourceFiles
  [ ] adminer
  [✓] examples
  [✓] images
  [✓] includes
  [✓] javascript
  [✓] sql
  [ ] sqlbuddy
  [✓] stylesheets
  [✓] temp
  [✓] templates
  [✓] amendcontent.php
  [✓] index.php
  [✓] licence.txt
  [✓] login.php
  [✓] phpinfo.php
```

Check All

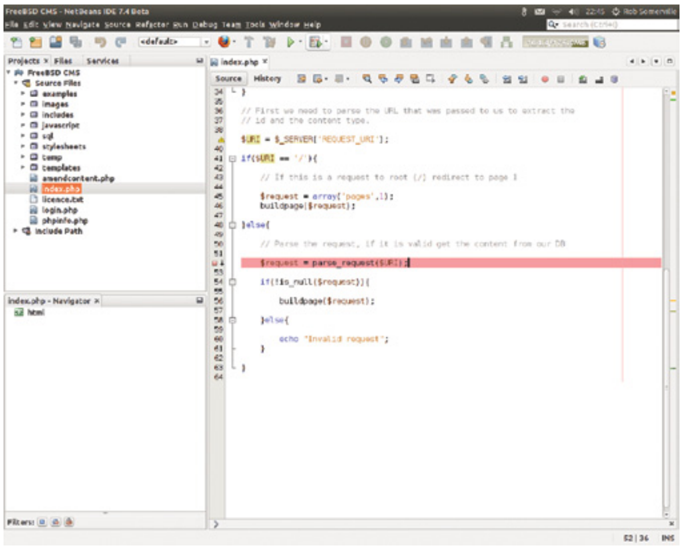
13 files (or more) selected. Selected files will be downloaded, existing files will be overwritten.

Project files will be automatically uploaded to the server every time you edit any of them.

Output - Remote Log: FreeBSD

```
/usr/home/dev/data
CWD /usr/home/dev/data
Directory successfully c
LIST
Here comes the directory
Directory send OK.
CWD /usr/home/dev/data
Directory successfully c
LIST
Here comes the directory
Directory send OK.
```

Figure 9. *Download the source tree – disable Adminer and sqlbuddy*


FreeBSD CMS - NetBeans IDE 7.4 Beta
File Edit View Navigate Source Refactor Run Debug Team Tools Window Help
Projects
Files
Services
index.php
Source
History
index.php - Navigator
html
Filters:

```
FreeBSD CMS
  Source Files
    examples
    images
    includes
    javascript
    sql
    stylesheets
    temp
    templates
    amendcontent.php
    index.php
    licence.txt
    login.php
    phpinfo.php
  Include Path
```

```
34  }
35
36  // First we need to parse the URL that was passed to us to extract the
37  // id and the content type.
38
39  $URI = $_SERVER['REQUEST_URI'];
40
41  if($URI == '/'){
42
43      // If this is a request to root (/) redirect to page 1
44
45      $request = array('pages',1);
46      buildpage($request);
47
48  }else{
49
50      // Parse the request, if it is valid get the content from our DB
51
52      $request = parse_request($URI);
53
54      if(!is_null($request)){
55
56          buildpage($request);
57
58      }else{
59
60          echo "Invalid request";
61      }
62
63  }
64
```

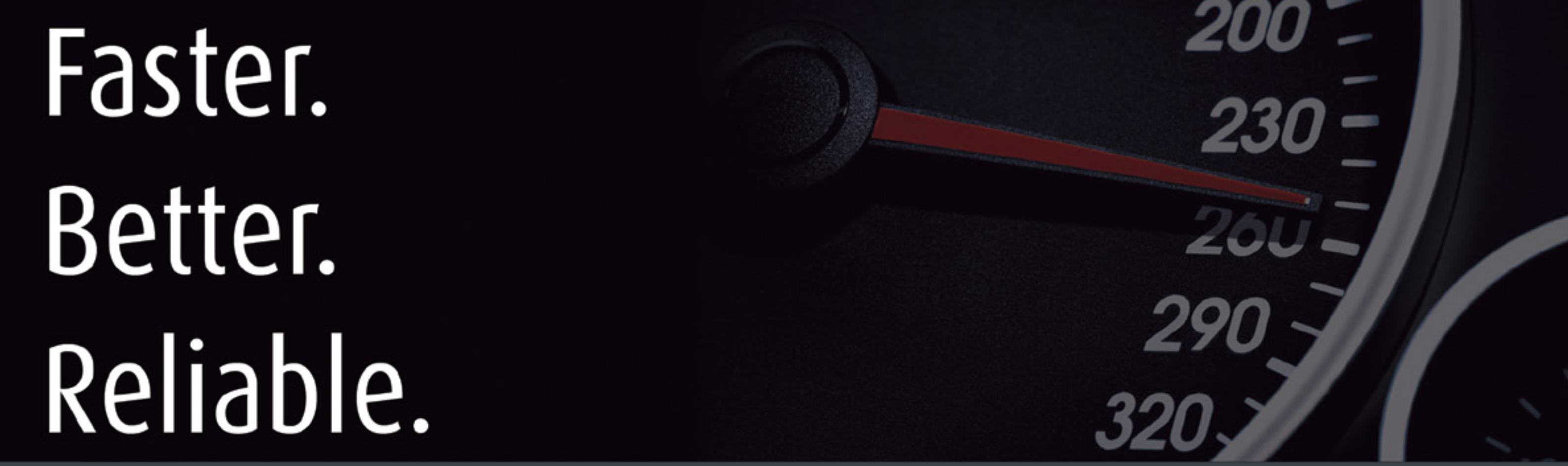
52 | 36 INS

Figure 10. *Load Index.php click on line 52 and start the debug session by pressing Ctrl F5*

Now follow the Figures (Figure 2 – 10). If all goes to plan, you should be able to step through your code by pressing F7, and interrogate variables by hovering over them e.g. `$request`. While debugging, the breakpoint line should change colour from pink to green. If it does not, there is some mis-communication between Netbeans and the server. See xdebug.log for further details.

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.



Faster.
Better.
Reliable.

Trusted by over 500 ISPs worldwide.

Hyper is the first multimedia cache fully developed in Brazil, by Taghos. With Hyper, ISPs can save on network bandwidth while increasing content-delivery speeds, resulting in end-customer satisfaction.

Features:

- 24x7x365 always-on support
- Active monitoring
- Automatic updates
- Appliance or license
- Easy deployment
- Configuration and reports via web interface

hyper

Remote Install
Using your hardware

Model	Traffic	RAM	Cache	SSD
T15	Up to 15 Mbps	8 GB	1x 1 TB	-
T50	Up to 50 Mbps	8 GB	2x 1 TB	-
T100	Up to 100 Mbps	8 GB	2x 1 TB	1x 160 GB
T150	Up to 150Mbps	16 GB	3x 2 TB	1x 160 GB
T300	Up to 300 Mbps	16 GB	5x 2 TB	1x 240 GB
T500	Up to 500 Mbps	32 GB	7x 2 TB	1x 480 GB
T1000	Up to 1 Gbps	64 GB	10x 1 TB	1x 480 GB
T2000	Up to 2 Gbps	96 GB	24x 1 TB	3x 480 GB
T3000	Up to 3 Gbps	128 GB	32x 1 TB	5x 480 GB

Visit us at **www.taghos.com** and start saving bandwidth today!

Unix Basics – for Security Professionals

Unix is the widely known multi-user and multitasking operating system that exists in many variants (e.g. Solaris, Linux, UX, AIX ...etc), and for serving mission critical server environments around the world.

What you will learn…

- How to provide a secure and reliable environment to the users
- How UNIX addresses the security challenges

What you should know…

- Unix basics
- Unix core components

Being a multi-user system, Unix has the responsibility to provide a secure and reliable environment to its users. Just like any other multi-user networking operating system, Unix also has two important Security Challenges to deal with, and they are:

1. Maintaining UNIX Internal security

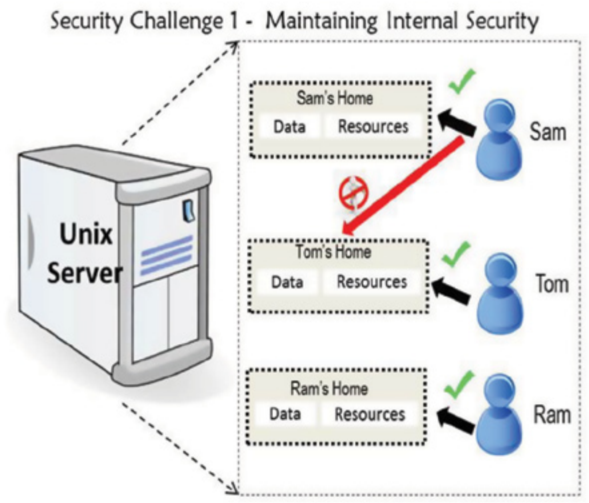


Figure 1. *Maintaining UNIX Internal security*

2. Defending UNIX Server Environment from External Threats

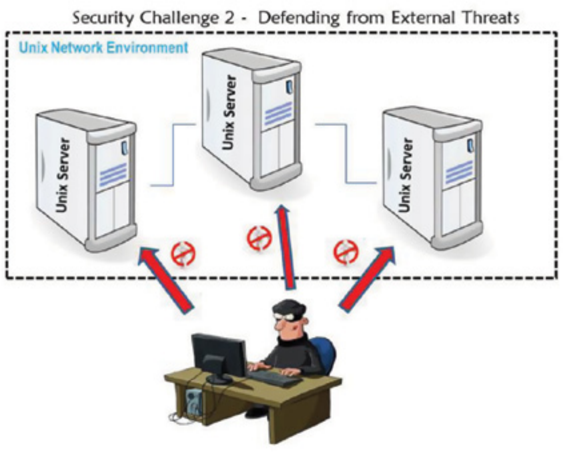


Figure 2. *Defending UNIX Server Environment from External Threats*

What can you expect from this article?

This article intends to provide the basics of Unix Operating systems while discussing how UNIX addresses the above security challenges. This is not a complete UNIX command by command tutorial, but rather a bird's eye view of overall Unix Operating system functions in simple terms. To begin, Unix-like operating systems are composed of several core components that are packaged and function together to deliver certain services to the Unix end-user. The diagram below gives you an overall picture of UNIX core components and their interconnection with other components.

Kernel

The source code of a Unix system that performs major operating system functions and also directly interacts with the server hardware with the help of sub-components like:

- Dev – Contains Device Drivers to control the hardware
- Sys – Handles key Operating system functions like memory management, process handling and system calls etc.

User Interface Environment

Users' interaction with the Unix operating system happens in several ways as classified below:

Shell Interface: Shell is a programmable command line interpreter which acts as a primary interface between the user and the Unix operating system. In the Modern Unix world there are several types of shell interfaces available, e.g. BASH, CSH, KSH, etc.

System and User Utilities – These are the tools which are provided for additional functionality of the Unix operating system, e.g. disk management tools like format, fdisk, etc.

Development Environment

UNIX provides built-in development tools to recreate the majority of the operating system from the source code, e.g. cc, as, ld, make, etc.

Key Functionalities of UNIX Operating systems

Unix Users & Groups

Now it's time to discuss 6 key functionalities of Unix that help us understand "How Unix addresses security challenges" as we discussed earlier in this article. Unix classifies its users into two categories, the first being super users who have complete privileges on the entire Unix operating system and the second being regular users who have privileges to access their own data and

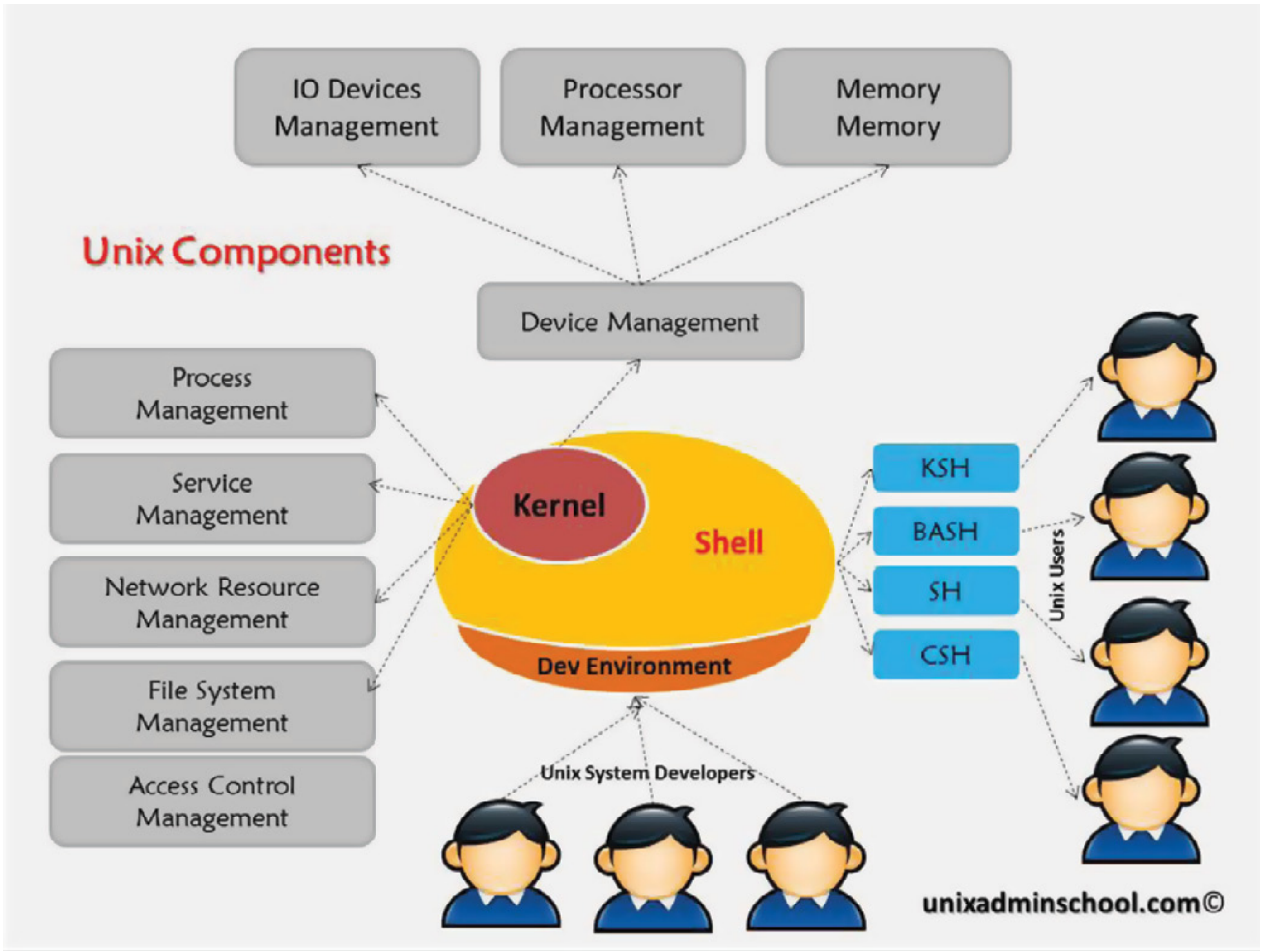


Figure 3. *UNIX core components*

resources only. Groups act as containers for users who require equal privileges on the same set of data and resources.

From the Security point of view, “Unused User accounts” is one of the areas that we continuously audit and disable if any unexpected user accounts are found. UNIX maintains its local user account database in three files: `/etc/passwd`, `/etc/shadow` and `/etc/group`. When working in an enterprise network environment, Unix is used to maintain a centralized user account database using NIS, NIS+ or LDAP.

Unix Allocates UserID (UID) to every User, and by default Super user accounts will have the UID of 0. For example, UNIX designates some users as system default users; they will be assigned with the Specific range of UIDs and normal users with a different range of UIDS. For example, in the Redhat Linux system default users will get UIDs < 500 whereas normal users will have a UID > 500. Here are a few steps to trace out unwanted/unexpected user accounts.

1. Look in `/etc/passwd` for new accounts in sorted list by UID:

```
# sort -nk3 -t: /etc/passwd | less
```

   Normal accounts will be there, but look for new, unexpected accounts, especially with UID < 500.
2. Also, look for unexpected UID 0 accounts:

```
# egrep ‘:0+:’ /etc/passwd
```

3. On systems that use multiple authentication methods:

```
# getent passwd | egrep ‘:0+:’
```

Unix Files, Directory and File Systems

In brief, files are used to store the data (in different formats like ascii format, binary format, etc.), whereas directories act as containers to group all related files in a single location, for easy maintenance of data. The File System is a structure that explains how the information is stored and retrieved from the Unix system.

Unix users access files for read, write and execute purposes. And Unix gives us the flexibility to assign these permissions to the file owners, groups and others individually.

Each file in Unix will have its own access control information, called Inode information, along with the actual data. Inode data (also referred to as file metadata) helps us to identify file type, file permissions for owner, group and others, file owner, file group, file size, file modification date, file modification time, etc. We can see the information with the command below:

```
# ls -l unixfile
drwxr-xr-x- 2 unix system 4096 Sep 27 23:38 unixfile
```

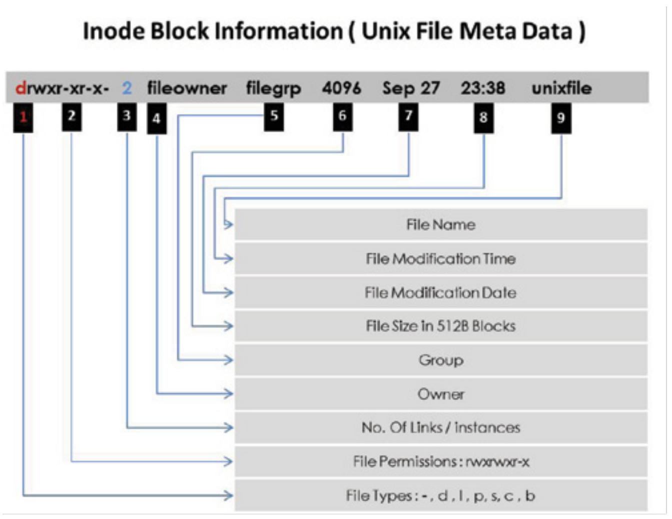

Inode Block Information (Unix File Meta Data)
drwxr-xr-x- 2 fileowner filegrp 4096 Sep 27 23:38 unixfile
1
2
3
4
5
6
7
8
9
File Name
File Modification Time
File Modification Date
File Size in 512B Blocks
Group
Owner
No. Of Links / Instances
File Permissions : rwxrwxr-x
File Types : -, d, l, p, s, c, b

Figure 4. *Inode Block Information*

In addition to regular access privileges, Unix also provides two additional ones – SUID and SGID, which allow users to run executable programs with owner and group permissions. From the security point of view, it is important to audit the system regularly to check if there are unexpected root owned files that are assigned with these additional privileges. To find unusual SUID root files, use the following command:

```
# find / -uid 0 -perm -4000 -print
```

and also look for files named with dots and spaces (“...”, the”.. “,”. “, and “ “) used to camouflage files:

```
# find / -name “ “ -print
# find / -name “.. “ -print
# find / -name “. “ -print
# find / -name “ “ -print
```

Unix Directory Structure

Unix maintains its entire information in a hierarchical tree form and the base of the tree is called the root (/) directory. Most Unix variants use this as the home directory of super user accounts. Unix classifies the files based on their purpose and places them under different sub-directories under “/” directory, the most well-known subdirectories being: `/bin` (user binaries); `/sbin` (system binaries); `/etc` (customized configuration files); `/dev` (device files); `/proc` (Active process information); `/var` (system Logs); `/tmp` (temporary files); `/opt` (optional programs); `/lib` (system libraries).

From the security point of view, we need to look into the system logs for suspicious events similar to

- Large number of Authentication failures via sshd, telnetd...etc
- Large number of RPC program logs showing with additional ascii codes
- Large number "Error" logs for – web servers, file servers...etc
- Unexpected application restarts or System Reboots, with truncated or disappeared system logs.

Every Unix variant has a security level in its kernel; the higher it is, the more secure the system is. Be aware that having a higher security level might cause performance degradation in the long-term. For example, in Linux, if you change the attribute of `/var/log/auth.log` to append mode as given below, then the intruder getting root privileges can't delete his root until he exclusively unsets the attribute. Here is a quick example to set attribute to auth.log:

```
# chattr +a auth.log
```

Unix Software Management and Patches

Almost all the Unix variants maintain their software components in the form of packages. A package is a collection of files and directories grouped together as per the System V interface definition. Once a package is developed and released for the installation, if there is any known potential problem found from the package, then the operating system vendor has to develop and release a fix, called a patch, for the problem. Sometimes patches are also used to provide a new feature or enhancement to a particular software package. Almost all Unix Variants have their own package and patch manager to perform regular package/patch installation, removal and update operations. For example, Linux variants use dpkg, rpm and yum. Regular patch management is very important for system security because every UNIX Operating System will have some unknown built-in security threats that are discovered over time. To run our Unix environment with the highest level of security, we always have to keep our security patches installed to up-to-date versions. Most OS variants will have mailing lists and/or other methods of informing users of important security updates. When installing the packages or patches, one key thing that we should remember is to make sure we are installing the right package (including version) and make sure none of the installed packages have been modified by an intruder to trick you to install their own packages. So how do we check the packages? Simple, just find the checksum for the package/patch, and compare the result with the original checksum provided with the original patch/package:

```
#md5sum samba-patch-x.x.x.rpm.bz2
67534a24ca89b76f5ae197ed171bd75e samba-patch-x.x.x.rpm.bz2
```

One can also check the gpg signature of the program tarball if present:

```
$ gpg --import samba-pubkey.asc
$ gunzip samba-version.tar.gz
$ gpg --verify samba-release.tar.asc
gpg: Signature made Tue 20 Nov 2007 07:12:04 PM CST using \
 DSA key ID 6568B7EA
gpg: Good signature from "Samba Distribution Verification Key \
 ‹samba-bugs@samba.org›"
```

[GEEKED AT BIRTH]

University of Advancing Technology
UAT
Learn. Experience. Innovate.

You can talk the talk.
Can you walk the walk?

[IT'S IN YOUR DNA]

LEARN:
Advancing Computer Science
Artificial Life Programming
Digital Media
Digital Video
Enterprise Software Development
Game Art and Animation
Game Design
Game Programming
Human-Computer Interaction
Network Engineering
Network Security
Open Source Technologies
Robotics and Embedded Systems
Serious Game and Simulation
Strategic Technology Development
Technology Forensics
Technology Product Design
Technology Studies
Virtual Modeling and Design
Web and Social Media Technologies

www.uat.edu > 877.UAT.GEEK

Please see www.uat.edu/fastfacts for the latest information about degree program performance, placement and costs.

Unix Processes and Services/Daemons

UNIX Process: Any executable program that is running in a Unix system, and consuming system resources like CPU/MEM/IO, is called a process. Unix will assign a Unique process ID (PID) and Process priority to every process during its initiation and will continue to monitor the process using the PID assigned. At any point in time the process will stay in any of the following states: running, waiting, sleeping and Zombie/Defunct (i.e. a completed child process without parent process). The `ps` command helps us to find the current active processes:

```
$ ps aux
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
timothy 29217 0.0 0.0 11916 4560 pts/21 S+ 08:15 0:00 pine
root 29505 0.0 0.0 38196 2728 ? Ss Mar07 0:00 sshd: can [priv]
can 29529 0.0 0.0 38332 1904 ? S Mar07 0:00 sshd: can@notty
USER = user owning the process
PID = process ID of the process
%CPU = It is the CPU time used divided by the time the
   process has been running.
%MEM = ratio of the process's resident set size to the
   physical memory on the machine
VSZ = virtual memory usage of entire process
RSS = resident set size, the non-swapped physical memory
   that a task has used
TTY = controlling tty (terminal)
STAT = multi-character process state
START = starting time or date of the process
TIME = cumulative CPU time
COMMAND = command with all its arguments
```

For security purposes, if you see any process that is unusual or unfamiliar, investigate in more detail using:

```
# lsof -p [pid]
```

This command shows all files and ports used by the running process. *Unix services*: These are the programs that initiate a set of processes to deliver specific operating system functionality, for example – print service, network service, back-up service ...etc. If the Services automatically start during the system startup and are running in the background mode without any user intervention, then those services are called system Daemons. To address security risks, we should be disabling any unused services that are running on the system and disable any related network ports. For example, in Linux you can see all the service information using the command:

```
#chkconfig -list
```

Unix Configuration in Network Environment

To configure Unix Systems in an Enterprise Network environment we have to deal with several components as described below:

- Functional Network Hardware – Ethernet Cards, Network Links, Network Speed, Network Duplex, etc.
- Functional Network Protocol Configuration – IP Addresses, Network Routes, Subnet masks, etc.
- Functional Network Service Configuration – Service Specific Configuration Files, e.g. httpd.conf, smb.conf, iptables.conf, ftp.conf, sshd.conf, etc.

Sample Commands from Linux:

`# ifconfig -a` to check the current available network interfaces
`# ethtool -i ethx` to check the network link speed and duplex settings
`# tcpdump` to check the network traffic from the network interface
`# netstat -rn` to check the network routes available in the systems
`# service <servicename>` start|stop to stop or start the network service

You can refer to my article – Linux Networking Troubleshooting (*http://gurkulindia.com/main/2012/11/redhat-enterprise-linux-networking-troubleshooting-quick-reference/*) if you want to know how to use these commands in real time. For general security, we should also occasionally:

- look for unusual port listeners: `# netstat -nap`
- get more details about running processes listening on ports: `# lsof -i`
- look for unusual ARP entries, mapping IP addresses to MAC addresses that aren't correct for the LAN: `# arp -a`

That's It. Now you know the overall functioning of Unix, and if you want expertise in any area, you can directly jump to that part and fine tune your understanding about that specific concept.

RAMKUMAR RAMADEVU

Ramkumar Ramadevu, is a well known author at unixadminschool.com who regularly writes about enterprise UNIX administration articles. Refer to his Unix Administration Bookshelf (http://unixadminschool.com/bookshelf/unixshelf.htm)

Professional services and solutions - Imperva, McAfee, HP, Tenable.
Penetration tests, Application Security, Managed Security Services (MSS).

Do as largest companies in Brazil, contact us!

www.ibliss.com.br info@ibliss.com.br +55 11 3255-3926

iBLISS
SEGURANÇA & INTELIGÊNCIA

Introduction to Unix Kernel

It is usually a source of wonderment to PC users that the whole of the Unix operating system is in one executable. Instead of a hodge-podge of DLL's, drivers, and various occasionally-cooperating executables, everything is done by the Unix kernel. When Unix was first introduced, the operating system was described as having a 'shell', or user interface, which surrounded a 'kernel' which interpreted the commands passed to it from the shell.

What you will learn…

- The Unix kernel functions
- How to improve Unix kernel processes

What you should know…

- Unix basics

With the passage of time and the advent of graphical window systems on Apollo and Sun computers in 1983, this model became a bit strained at the edges. However, it still provides a useful mental image of the system, and window systems can be thought of as a 'candy coat' around the shell. In fact, it isn't just X-windows, which has a direct path to the kernel, since TCP/IP also falls into this category.

The following is not intended to be an exhaustive treatise on the inner workings of the Unix kernel, nor is it specific to any particular brand of Unix. It is essential to broadly understand certain important functions of the kernel before we can take advantage of some of its features and improve the way in which it handles our processes. The Unix kernel possesses the following functions, much of which is of interest to us, in pursuit of this goal:

System calls

All of the most basic operating system commands are performed directly by the kernel. These include:

- `open()`
- `close()`
- `dup()`
- `read()`
- `write()`
- `fcntl()`
- `ioctl()`
- `fork()`
- `exec()`
- `kill()`

Since the above commands are executed by the kernel, the 'C' compiler doesn't need to generate any actual machine code to perform the function. It merely places a 'hook' in the executable, which instructs the kernel where to find the function. Modern third party compilers, however, are ported to a variety of operating systems, and will generate machine code for a dummy function, which itself contains the 'hook'.

Process scheduling and control

The kernel determines which processes will run, when and for how long. We will examine this mechanism in detail later.

Networking

That which became networking was originally designed so that processes could communicate. It is this ability

NET OPEN SERVICES IS AN APPLICATION HOSTING COMPANY FOCUSED ON OPEN SOURCE APPLICATIONS MANAGEMENT IN HIGH AVAILABILITY ENVIRONMENT.

NET OPEN SERVICES IS PROUD TO PROVIDE A HIGH QUALITY SERVICE TO OUR CUSTOMERS SINCE 10 YEARS.

OUR EXPERTISE INCLUDES:

- **CLOUD COMPUTING, PUBLIC, PRIVATE AND HYBRID CLOUD MANAGEMENT (OPENSTACK, CLOUDSTACK, RED HAT ENTERPRISE VIRTUALIZATION)**
- **REMOTE MONITORING AND MANAGEMENT 24/7**
- **NETWORKING AND SECURITY (OPEN BSD, IP TABLE, CHECKPOINT, CISCO,...)**
- **OS AND APPLICATION MANAGEMENT (FREE BSD, OPEN BSD, SOLARIS, UNIX, LINUX, AIX, MS WINDOWS)**
- **DATABASE MANAGEMENT (ORACLE, MYSQL, CASSANDRA, NOSQL, MS SQL, SYBASE...)**
- **MANAGED HOSTING IN CARRIER CLASS DATA CENTERS**
- **DISASTER RECOVERY**

Net Open SERVICES

WE PROVIDE SERVICES IN EVERY STEP OF THE PROJECT LIFE, DESIGN, DEPLOYMENT, MANAGEMENT AND EVOLUTIONS. **NETOPENSERVICES** TEAM INCLUDES EXPERIENCED LEADERS AND ENGINEERS IN THE INTERNET SERVER INDUSTRY.

OUR TEAM HAS **15 YEARS OF EXPERIENCE** IN DEVELOPING INTERNET INFRASTRUCTURE-GRADE SOLUTIONS AND PROVISIONING INTERNET DATACENTERS AND GLOBAL SERVICE NETWORKS TOGETHER.

WE OFFER EXCEPTIONAL HARDWARE SUPPORT AS SOFTWARE SUPPORT ON UNIX/LINUX AND OPEN SOURCE APPLICATION. **NETOPENSERVICES** DELIVERS THESE CUSTOM-BUILT LINUX AND UNIX SERVERS, AS WELL AS PRECONFIGURED SERVERS AND SCALABLE STORAGE SOLUTIONS, TO OUR CUSTOMERS. WE ALSO OFFER CUSTOM DEVELOPMENT AND ADVANCED-LEVEL UNIX/LINUX CONSULTING SOLUTIONS.

WWW.NETOPENSERVICES.COM • CONTACT@NETOPENSERVICES.COM

to pass information quickly and efficiently from one running process to another that makes the Unix operating system uniquely capable of multidimensional operation. All of the most important communication commands are system calls.

- `socket()`
- `connect()`
- `bind()`
- `listen()`
- `accept()`
- `send()`
- `recv()`

Device drivers for all supported hardware devices

Unlike other operating systems, where device drivers are separate programs individually loaded into memory, the Unix kernel inherently contains all of the machine's drivers. Contrary to what may be supposed, the entries in the /dev directory are not drivers, but rather are just access points into the appropriate kernel routine. We will not concern ourselves unduly with the Unix device drivers, as they are outside the scope of this article.

Anatomy of a process

Single-threaded:

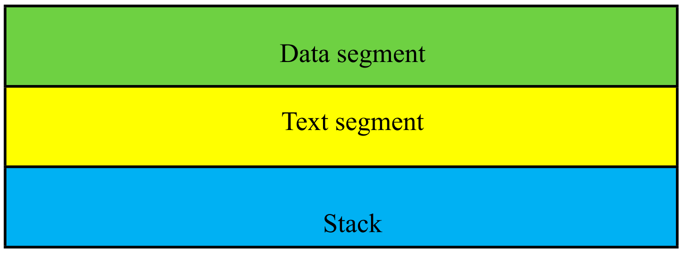


Figure 1. *Anatomy of the process – single-threaded*

Multi-threaded:

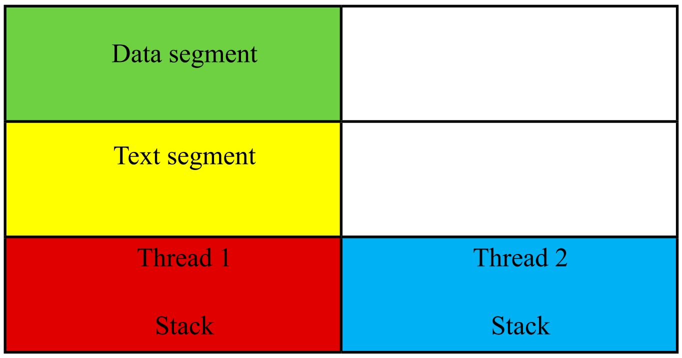


Figure 2. *Anatomy of the process – multi-threaded*

When an executable is invoked, the following events occur, though not necessarily in this order.

Process Loading

The loader

- Fetches the executable file from the disk
- Allocates memory for all of the global variables and data structures ('the data segment') and loads the variables into that area of memory.
- Loads the machine code of the executable itself ('the text segment') into memory. With demand-paged executables this is not strictly the case, as the amount of code actually loaded into memory is several 4k pages. The remainder is put into the swap area of the disk.
- Searches the header portion of the executable for any dynamically-linked libraries or modules.
- Checks to see if these are already loaded and, if not, the modules are loaded into memory. Otherwise, their base addresses are noted.
- Makes available the base addresses of dynamically-linked modules/libraries to the process.
- Allocates an area of memory for the stack. If the process is multi-threaded, a separate stack area is allocated for each thread.

The kernel

- Sets the program counter to the first byte of executable code.
- Allocates a slot in the process table to the new process.
- Allocates a process ID to the new process.
- Allocates table space for any file descriptors.
- Allocates table space for any interrupts.
- Sets the 'ready to run' flag of the process.

All of the above resources, allocated to a given process, constitute the 'context' of a process. Each time the kernel activates a new process, it performs a context switch by replacing the resources of the previously running process with those of the current one. At this point, the scheduling algorithm takes over.

The Process Scheduling algorithm

While a process is running, it runs in one of two modes:

Kernel mode

All system calls are executed by the kernel and not by machine code within the process. Kernel mode has one very desirable characteristic, and that is the fact that system calls are atomic and hence, cannot be interrupted. One of the most important factors in writing code for high-performance applications, is to ensure that your process executes as much in kernel mode as possible.

This way you can guarantee the maximum CPU time for a given operation.

Kernel threads, such as pthreads, run in kernel mode. It is sometimes worth using a thread, even when doing so doesn't constitute parallel operation, purely to get the advantage of running in kernel mode. If an interrupt occurs, while in this mode, the kernel will log the signal in the interrupt table, and examine it after the execution of the current system call. Only then will the signal be actioned.

User mode

The ordinary machine code, which makes up much of the executable, runs in user mode. There are no special privileges associated with user mode, and interrupts are handled as they arrive. It may be seen that, during the time that a process runs, it is constantly switching between kernel mode and user mode. Since the mode switch occurs within the same process context, it is not a computational burden.

Scheduling

A Unix process has one of the following states:

- Sleep
- Run
- Ready to Run
- Terminated

The scheduling algorithm is a finite-state machine, which moves the status of the process between states, depending on certain conditions. Basically, what happens is this. A process begins to execute. It runs until it needs to perform I/O, then, having initiated the I/O, puts itself to sleep. At this point, the kernel examines the next process table slot and, if the process is ready to run, it enables its execution. If a process never performs I/O, such as processes which perform long series of floating point calculations, the kernel permits it to only run for a fixed time period, of between 20 and 50 milliseconds, before pre-empting it, and enabling the next eligible process.

When the time comes for a process to run, an additional algorithm determines the priority of one process over another. The system clock sends the kernel an interrupt once per second, and it is at this time that the kernel calculates the priorities of each process. Leaving aside the user-level priority weighting, determined by 'nice' the kernel determines priority based on several parameters, of which the following are significant:

- How much CPU time the process has previously used
- Whether it is waking up from an I/O wait or not
- Whether it is changing from kernel mode to user mode or not

Swapping and Paging

Of course, the execution of a process is never that straightforward. Only a portion of the code is loaded into memory, meaning that it can only run until another page needs to be fetched from the disk. When this occurs the process generates a 'page fault,' which causes the pager to go and fetch the appropriate page. A similar situation occurs when a branch instruction is executed, which takes the execution point to a page other than those stored in memory. The paging mechanism is fairly intelligent and contains algorithms similar to those found in CPU machine instruction pipeline controllers and tries to anticipate branch instructions and pre-fetch the anticipated page, more or less successfully, depending on the code structure.

Although there are certain similarities, paging, which is a natural result of process execution, should not be confused with swapping. If the number of processes grows to the extent that all available memory becomes used up, the addition of another process will trigger the swapper and cause it to take a complete process out of memory, and place it in the swap area of the disk. This is, computationally, an extremely expensive operation. The entire process, together with its context, has to be written to disk, then, when it is permitted once again to run, another process must be swapped out to make space for it to be reloaded into memory.

So, what does all of this have to do with Performance?

It may be seen from the above that processes, which are designed for performance-critical applications, should avoid doing physical I/O until it is absolutely necessary in order to maximize the amount of contiguous CPU time. If it is at all possible, all of the I/O operations should be saved up until all other processing has completed and then be performed in one operation, preferably, by a separate thread.

As far as threads are concerned, let us consider what happens, when we launch a number of threads, to perform some tasks in parallel. First, the threads are each allocated a separate stack, but are not allocated a separate process table slot. This means that, although there are several tasks executing in parallel, this only occurs during the active time of that slot. When the kernel preempts the process, execution stops. Multi-threading will not give your application any more system resources. Further, if we consider a situation where we have 100 processes running on a machine and one of them is ours, then we would expect to use 1% of the CPU time. However, if 25 of them are ours, we would be eligible to use 25% of the CPU time.

Thus, if an application can split itself into several processes, running concurrently, then, quite apart from the obvious advantages of parallelism, we will capture more of the machine's resources simply because each child process occupies a separate process table slot. This also helps when the kernel assigns priorities to processes. Even though we may be penalized for using a lot of CPU time, the priority of each process is rated against that of other processes. If many of these belong to one application, then even though the kernel may decide to give one process priority over another, the application, as a whole, will still get more CPU time.

Additionally, if we are running on a multi-processor machine, then we can almost guarantee to be given a separate CPU for each child process. The kernel may juggle these processes over different CPU's, as a part of its load-balancing operations, but each child will still have its own processor. The incorporation of the above techniques into our software architecture forms the cornerstone of multi-dimensional programming.

Process Scheduling, in Summary

- Each child process gets a CPU different to that used by the parent.
- The more processes contribute to the running of your application, the more CPU time it will get.
- Multi-threading creates multiple execution paths within one process table slot. It may permit parallel execution paths, but it will not get the application more CPU time, or a new CPU.

Therefore:

- Find parallelism within your application. This will make your software run more efficiently.
- Employ multi-threading where it is not possible to fork a separate process, or where you need to refer to global information, as in the parent process.
- Having decided how the children will communicate the data back to the parent, launch a separate child process for every possible parallel function, to gain the maximum CPU time.

System calls

fork()

Under pre-Unix operating systems, starting a process from within another process was traditionally performed as a single operation. One command magically placed the executable into memory, and handed over control and ownership to the operating system, which made the new process run. Unix doesn't do that.

Each process has a hierarchical relationship, with its parent, which is the process which brought it to life, and with its child or children which, in turn, are processes which it, itself, started. All such related processes are part of a `process group`. If a `kill()` signal is sent to the parent of the process group, it will propagate to the child processes. Unix also has the concept of a 'session' which, essentially, can be thought of as comprising all of the process groups associated with a login, or TCP/IP connection.

The basic mechanism that initiates the birth of a new process is `fork()`. The `fork()` system call makes a running copy of the process which called it. All memory addresses are re-mapped, and all open file descriptors remain open. Also, file pointers maintain the same file position in the child as they do in the parent. Consider the following code fragment:

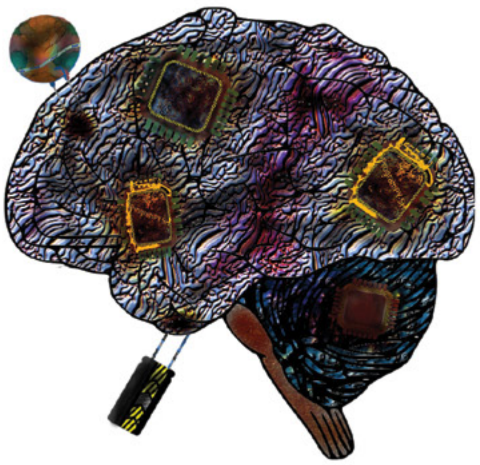
```
pid_t pid;

    switch((pid = fork()){
        case -1:
            printf("fork failed\n");
        break;
        case 0:
            printf("Child process running\n");
            some_child_function();
        break;
        default:
            printf("Parent process executes this code\n");
        break;
    }
```

At the time that the `fork()` system call is called there is only one process in existence, that of the expectant parent. The local variable pid is on the stack, probably uninitialised. The system call is executed and, now, there are two identical running processes both executing the same code. The parent and the new child process both simultaneously check the variable pid, on the stack. The child finds that the value is zero and knows, from this, that it is the child. It then executes `some _ child _ function()` and continues on a separate execution path.

The parent does not see zero, so it executes the 'default' part of the `switch()` statement. It sees the process ID of the new child process, and drops through the bottom of the `switch()`. Please note that if we do not call

Current Projects by Devyn Collier Johnson <DevynCJohnson@Gmail.com>



Neobot is one of several Betabots. All Betabots are advanced chatbots that use the Pysh engine and read Xaiml files. The Betabots, Pysh, and Xaiml are still developing technolgies made by Devyn Collier Johnson. Neobot and the Xaiml specification can be found here https://launchpad.net/neobot

I make many wallpapers for free. Come check them out http://gnome-look.org/usermanager/search.php?username=DevynCJohnson

I write articles for Linux.org. Come check out the place and enjoy the site. Linux.org offers forums and tutorials as well as informative articles.

Want me as your writer? Email me your name, website/company, email address, a list of desired articles (topic,article size, and so on), and payment method/amount. Feel free to ask me further questions. To learn more about me, go to this site (https://launchpad.net/~devyncjohnson-d).

I am also the beta-tester for Wallch and SuperTux.

a different function in the case 0: section of the switch, both parent and child will continue to execute the same code, since the child will also drop through the bottom of the `switch()`.

Programmers who know little about Unix will have a piece of folklore rattling around in their heads which says 'a `fork()` is expensive. You have to copy an entire process in memory, which is slow, if the process is large'. This is true, as far as it goes. There is a memory-to-memory copy of that part of the parent, which is resident in memory, so you may have to wait a few milliseconds. However, we are not concerned with trivial processes whose total run time is affected by those few milliseconds. We are dealing exclusively with processes whose run times are measured in hours, so we consider a one-time penalty of a few milliseconds to be insignificant.

When a parent forks a child process on a multi-processor machine, the Unix kernel places the child process onto its own separate CPU. If the parent forks twelve children on a twelve CPU machine, each child will run on one of the twelve CPU's. In an attempt to perform load-balancing, the kernel will shuffle the processes around the CPU's, but, basically, they will remain on separate processors.

The `fork()` system call is one of the most useful tools, for the full utilisation of a multi-processor machine's resources, and it should be used whenever one or more functions are called, which can proceed their tasks in parallel. Not only is the total run time reduced to that of the longest-running function, but each function will execute on its own CPU.

vfork()

There is a BSD variant of `fork()`, which was designed to reduce the memory usage overhead associated with copying, possibly, a huge process in memory. The semantics of `vfork()` are exactly the same as those of `fork()`, but the operation is slightly different. `Vfork()` only copies the page of the calling process which is currently in memory, but, due to a bug (or feature), permits the two processes to share the same stack. As a result, if the child makes any changes to variables local to the function which called `vfork()`, the changes will be visible to the parent. Knowledge of this fact has enabled experienced programmers to make use of the advantages of `vfork()`, while avoiding the pitfalls. However, far more subtle bugs also exist, and most Unix vendors recommend that `vfork()` only be used, if it is immediately followed by an `exec()`.

exec()

The original thinking behind `fork()`, was that its primary use would be to create new processes, not just copies of the parent process. The `exec()` system call achieves this by overlaying the memory image of the calling process with the new process. There is a very good reason for separating `fork()` and `exec()`, rather than having the equivalent of VMS's `spawn()` function, which combines the two. That reason is because it is sometimes necessary, or convenient, to perform some operations in between `fork()` and `exec()`. For example, it may be necessary to run the child process as a different user, like root, or to change directory, or both. There is, in fact, no such call as `exec()`, but there are two main variants, `execl()` and `execv()`. The semantics of `execl()` are as follows:

```
execl(char *path, char *arg0, char *arg1…char *argn, (char
   *) NULL)
execv(char *path, char *arg0, char **argv)
```

It may be seen, that the principal difference between the two variants, is that, whereas the `execl()` family takes a path, followed by separate arguments, in a NULL terminated, comma-separated list, the `execv()` variants take a path, and a vector, similar to the `argv[]` vector, passed to a `main()` function.

The first variant of `execl()` and `execv()`, adds an environment vector to the end of the argument list:

```
execle(char *path, char *arg0, .…char *argn, (char *)
   NULL, char **envp)
execve(char *path, char *arg0, char **argv, char **envp)
```

The second variant replaces the 'path' argument, with a 'file' argument. If this latter contains a slash, it is used as a path. Otherwise, the PATH environment variable of the calling process is used to find the file.

```
execlp(char *file, char *arg0, .…char *argn, (char *) NULL,
   char **envp)
execvp(char *file, char *arg0, char **argv, char **envp)
```

We can now combine `fork()` and `exec()` to execute lpr from the parent process in order to print a file:

```
    pid_t pid;
    switch((pid = fork()){
        case -1:
            printf("fork failed\n");
        break;
        case 0:
            printf("Child process running\n");
            execl("/usr/ucb/lpr", "lpr", "/tmp/file", (char
   *) NULL);
        break;
        default:
            printf("Parent process has executed lpr to
   print a file\n");
        break;
    }
```

The above code only has one problem. If the parent process quits, the child process will become an orphan and be adopted by the 'init' process. When lpr has run to completion, it will become a zombie process and waste a slot in the process table. The same happens if the child prematurely exits, due to some fault.

There are two solutions to this problem. We execute one of the `wait()` family of system calls. A waited-for child does not become a zombie, but the parent must suspend processing, until the child terminates, which may or may not be a disadvantage. There are options which allow processing to continue during the wait, but the parent needs to poll `waitpid()`, which makes our second solution, described below, a much better option.

If we are waiting for a specific process, the most convenient call is to `waitpid()`. The synopsis of this call is:

```
pid_t waitpid(pid_t pid, int *status, int options);
```

The call to `waitpid()` returns the process ID of the child for which we are waiting, whose process ID is passed in as the first argument, 'pid'. The second argument, 'status', is the returned child process exit status and 'options' is the bitwise-OR of the following flags:

WNOHANG: prevents `waitpid()` from causing the parent process to hang, if there is no immediate return.

WNOWAIT: keeps the process, whose status is returned, in a waitable state, so that it may be waited for again.

The options flags are of no use to us, so we set them to zero. The status word, however, provides useful information on how our child terminated, and can be decoded with the macros, as described in the man page for 'wstat'.

```
pid_t pid;
int status;

    switch((pid = fork()){
        case -1:
            printf("fork failed\n");
        break;
        case 0:
            printf("Child process running\n");
            execl("/usr/ucb/lpr", "lpr", "/tmp/file", (char
   *) NULL);
        break;
        default:
            printf("Parent process has executed lpr to
   print a file\n");
 if(waitpid(pid, &status, 0) == pid){
      printf("lpr has now finished\n");
 }
        break;
    }
```

If we don't wish to poll `waitpid()` repeatedly, but need to do other processing while the child process goes about its business, then we need to effectively disown the child process. As soon as the child has successfully forked, we must disassociate it from the process group of the parent.

Process groups and sessions are discussed at the beginning of the `fork()` section but, to save you the trouble of looking, a process group is headed by the parent process whose process ID becomes the group's process group ID. All children of the parent then share this process group ID. The disowning of a child process is accomplished by executing the system call `setpgrp()`, or `setsid()`, (both of which have the same functionality) as soon as the child is forked. These calls create a new process session group, make the child process the session leader, and set the process group ID to the process ID of the child. The complete code is as below:

```
pid_t pid;

    switch((pid = fork()){
        case -1:
            printf("fork failed\n");
        break;
        case 0:
 if(setpgrp() == -1){
                printf("Can't set pgrp\n");
            }
printf("Independent child process running\n");
            execl("/usr/ucb/lpr", "lpr", "/tmp/file", (char
   *) NULL);
        break;
        default:
            printf("Parent process has executed lpr to
   print a file\n");
        break;
    }
```

open() close() dup() read() write()

These system calls are primarily concerned with files but, since Unix treats almost everything as a file, most of them can be used on any byte-orientated device, including sockets and pipes.

int open(char *file, int how, int mode)

`open()` returns a file descriptor to a file, which it opens for reading, writing or both. The 'file' argument is the file name, with or without a path, while 'how' is the bitwise-OR of some of the following flags defined in fcntl.h:

O_RDONLY	Read only
O_WRONLY	Write only
O_RDWR	Read/write
O_TRUNC	Truncate on opening
O_CREAT	Create if non-existent

The 'mode' argument is optional and defines the permissions on the file, using the same flags as chmod.

Int close(int fd)

Closes the file, which was originally opened with the file descriptor fd.

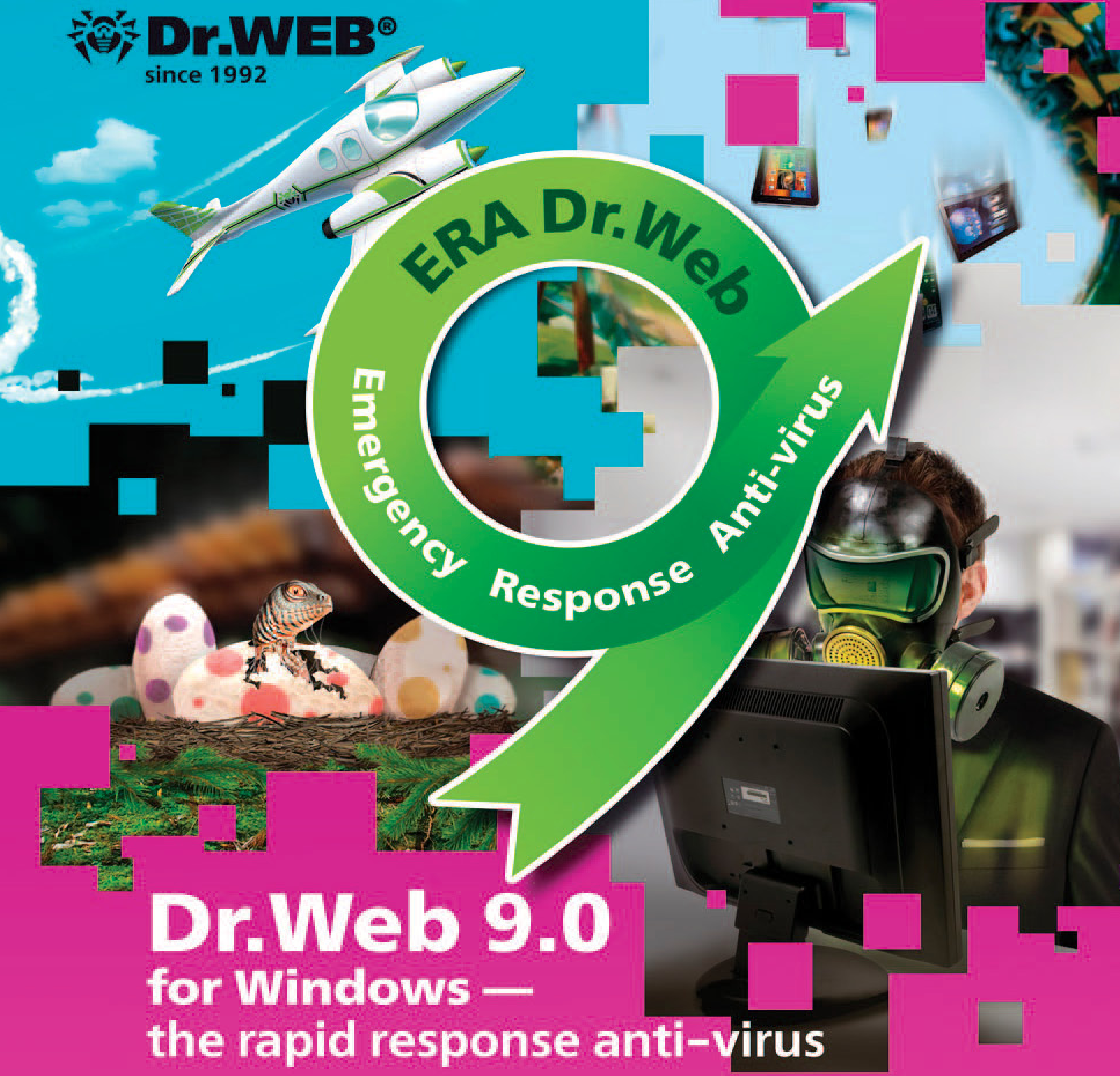
Int dup(int fd)

Returns a file descriptor, which is identical to what passed in as an argument, but with a different number. This call seems fairly useless at first glance, but, in fact, it permits some powerful operations like bi-directional pipes, where we need a pipe descriptor to become a standard input or output. Also, client-server systems need to listen for incoming connections on a fixed socket descriptor while handling existing connections on different descriptors.

MARK SITKOWSKI

Mark Sitkowski Design Simulation Systems Ltd http://www.designsim.com.au. Consultant to Forticom Security http://www.forticom.com.au

Dr.WEB®
since 1992



Dr.Web 9.0

for Windows — the rapid response anti-virus

1. Reliable protection against the threats of tomorrow
2. Reliable protection against data loss
3. Secure communication, data transfer and Internet search

www.drweb.com
Free 30-day trial: https://download.drweb.com
New features in Dr.Web 9.0 for Windows: http://products.drweb.com/9
FREE bonus — Dr.Web Mobile Security: https://download.drweb.com/android

OpenBSD 5.4

as a Transparent HTTP/HTTPS Proxy

In this article, we are going to build a firewall using OpenBSD 5.4 embedded with a transparent proxy that disallows some URLs as as a blacklist. It is not helpful to install squid for that. Relayd did the trick with the following bonus: HTTPs inspection!

What you will learn…

- How to configure Relayd for URL Blocking with https inspection
- How to use and understand Packet Filter

What you should know…

- Unix commands
- The basics of TCP/IP
- Configure OpenBSD network

Considering the network on Figure 1. To begin, please read the following man pages: PF.CONF(5), PFCTL(8), RELAYD.CONF(5), RELAYCTL(8), and SSL(8). It is essential to have network cards (`/etc/hostname.xxx`), gateway (`/etc/mygate`), and DNS resolver (`/etc/resolv.conf`) configured before starting this How-To.

What do we want to achieve?

Block «File Hosting» websites like 1fichier.com, uptobox.com, mega.co.nz …

All the urls we want to block are located in a file `/etc/filehosting`, as a blacklist.

Here is a sample for the file `/etc/filehosting`:

```
mega.co.nz/
uploaded.net/
uptobox.com/
```

Enable IPv4 Routing

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

In order to keep this setting at reboot enter the following:

```
echo "net.inet.ip.forwarding=1 » >> /etc/sysctl.conf
```

Interface Group

By default «`em0`» is part of egress group, and it is the interface connected to the Internet.

At this point we want to add em1 to the lan group, so we do the following:

```
/sbin/ifconfig em1 group lan
```

And to keep this setting at reboot:

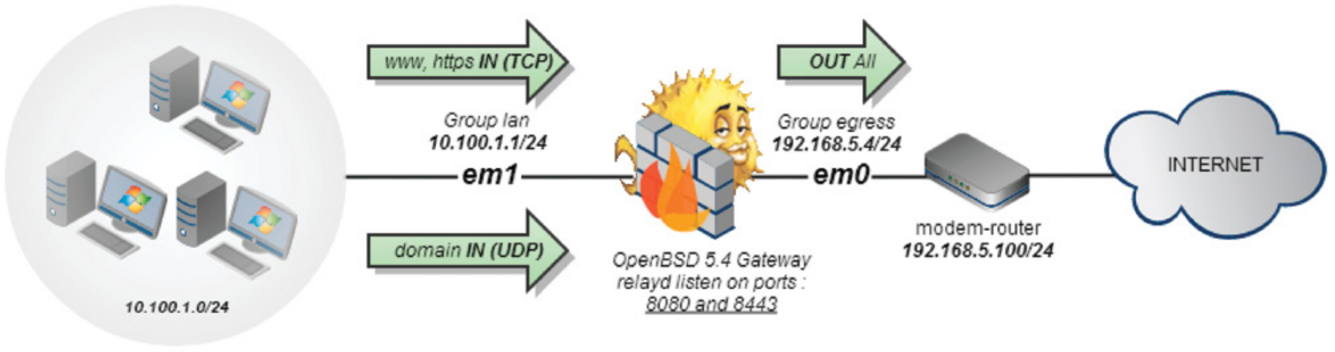


Figure 1. *A network diagram*

```
echo "!/sbin/ifconfig em1 group lan" >> /etc/hostname.em1
```

Packet Filtering

As it shows in the network diagram, we want to allow our workstations to use only www, https and domain resolution (Bonus: ping ;-).

By default, PF is enabled and here is the `/etc/pf.conf` ruleset:

```
# We declare bad hosts (some RFC like 1918…)
    mydns={8.8.8.8, 8.8.4.4}
martians="{ 127.0.0.0/8, 192.168.0.0/16, 172.16.0.0/12,
    10.0.0.0/8, \
    169.254.0.0/16, 192.0.2.0/24, 0.0.0.0/8, 240.0.0.0/24
    }"

# We don't need to load fingerprints
set fingerprints "/dev/null"

# No filters on loopback
set skip on lo

# NAT
match out on egress inet from lan:network to any nat-to
    egress
# Normalize packets
match in all scrub (no-df max-mss 1440)

# Policy: we block all and log
block log all
# Protection antispoof
antispoof for {egress,lan}
# We deny bad hosts
block in quick on egress from $martians
# We trust out on WAN
pass out on egress
# Redirect www traffic from our lan to relayd on port 8080
pass in quick inet proto tcp from lan:network to any port
    www \
    divert-to localhost port 8080
# Redirect https traffic from our lan to relayd on port
    8443
pass in quick inet proto tcp from lan:network to any port
    https \
    divert-to localhost port 8443
# We allow our network to use Google DNS resolution
pass in on lan inet proto udp from lan:network to $mydns
    port domain
# We allow pings
pass in on lan inet proto icmp from lan:network to any
    icmp-type echoreq
```

Keep FreeBSD Free!

The FreeBSD™ FOUNDATION

Support FreeBSD by donating to The FreeBSD Foundation

To find out more, please visit our Web site:
www.freebsdfoundation.org

Load PF ruleset:

```
 /sbin/pfctl -vf /etc/pf.conf
```

Relayd: url filtering for http/https

Create CA key and Certificate:

```
openssl req -x509 -days 365 -newkey rsa:2048 -keyout /etc/
   ssl/private/ca.key -out /etc/ssl/ca.crt
```

I chose «testing _ relayd» as a password. You will need it in the relayd.conf file, and the «ca.crt» needs to be installed on all the computers on the network (lan).

Create an SSL server key and certificate for 127.0.0.1:

```
openssl genrsa -out /etc/ssl/private/127.0.0.1.key 2048
```

Generate a Certificate Signing Request (CSR):

```
openssl req -new -key /etc/ssl/private/127.0.0.1.key \
            -out /etc/ssl/private/127.0.0.1.csr
```

Sign the key yourself:

```
openssl x509 -sha256 -req -days 365 \
            -in /etc/ssl/private/127.0.0.1.csr \
            -signkey /etc/ssl/private/127.0.0.1.key \
            -out /etc/ssl/127.0.0.1.crt
```

The /etc/relayd.conf should say:

```
http protocol "no_ssl" {
   return error
   label "File Hosting Websites is banned !"
   request url filter file "/etc/filehosting"
}

http protocol "with_ssl" {
   return error
   label "File Hosting Websites are banned !"
   request url filter file "/etc/filehosting"
   ssl ca key "/etc/ssl/private/ca.key" password "testing_
   relayd"
   ssl ca cert "/etc/ssl/ca.crt"
}

relay "no_ssl_proxy" {
   listen on 127.0.0.1 port 8080
   protocol "no_ssl"
   forward to destination
}

relay "with_ssl_proxy" {
   listen on 127.0.0.1 port 8443 ssl
   protocol "with_ssl"
   forward with ssl to destination
}
```

Start relayd:

```
echo relayd_flags= >> /etc/rc.conf.local
/etc/rc.d/relayd start
```

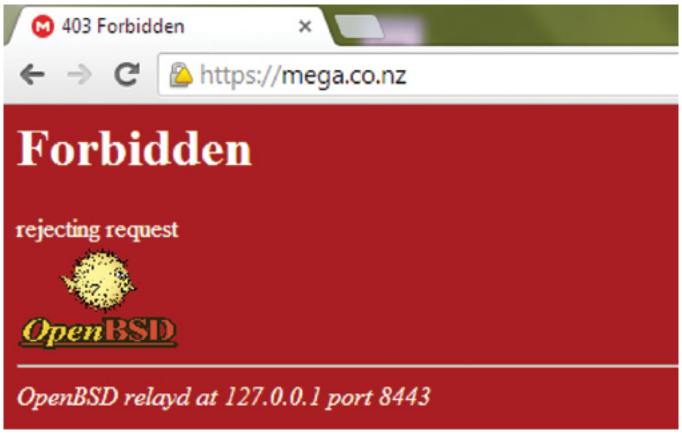
Load Relayd configuration:

```
/usr/sbin/relayctl load /etc/relayd.conf
```

Verify that relayd listen on 8080 and 8443:

```
/usr/bin/netstat -anf inet | grep 127.0.0.1.8     # This
   will give the following:
tcp        0       0  127.0.0.1.8443      *.*          LISTEN
tcp        0       0  127.0.0.1.8080      *.*          LISTEN
```

Test the url filtering on a workstation using the Chrome browser. To have a nicer Forbidden page, you can change the «label» value in /etc/relayd.conf to:



403 Forbidden

https://mega.co.nz

Forbidden

rejecting request

OpenBSD relayd at 127.0.0.1 port 8443

Figure 2. *A sample forbidden page*

```
label "<img src='http://www.openbsd.org/art/puffy/
   puflogv100X65.gif'/>"
```

WESLEY MOUEDINE ASSABY

Wesley MOUEDINE ASSABY lives in Reunion island, near Mauritius, works as network administrator at AISE-INFORMATIQUE (http://www.aise.re) where Wesley MOUEDINE ASSABY installs some firewalls (Soekris appliances), mail server, all using OpenBSD system. Wesley MOUEDINE ASSABY used it since 2007 (version 4.1), it became Wesley's passion.

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

FreeBSD mall

For even MORE items visit our website today!
www.FreeBSDMall.com

GhostBSD: A User-friendly, Light-weight BSD Alternative

GhostBSD is an open source desktop operating system based on FreeBSD which aims for a secure, user-friendly experience out of the box. GhostBSD comes with most common software choices already configured, giving the user a solid BSD installation out of the box.

As a long time BSD user, my search for a distribution has spanned the course of over 15 years. I started to use FreeBSD back in 1997 or 1998 when version 2.2 came out. Getting FreeBSD running on a computer or server was a work intensive experience that involved a lot of fine tuning and time with my nose in a manual. I remember expending hours and hours in front of my computer to get everything working, but at the end it was something awesome.

As FreeBSD didn't have good options for Desktop environments, with the coming years I started to use Windows and Linux. They both worked great and were very impressive to see on a desktop system. Windows offered a lot of functionality out of the box, but had stability issues and was a relatively bloated operating system. Linux also had their own system stability issues early on, but the desktop environment options were nice. Stability has improved greatly over the years, but due to the decentralized nature of those distributions, they tend not to be as cohesive and stable as FreeBSD operating systems and require the same amount of work in most cases.

When gnome came out for FreeBSD, it started to make the idea of a nice desktop installation possible. I remember using Gnome 1.4 for some time and then jumping to the famous Gnome 2. At that time, it still took a lot of work to get the desktop environments up and running. Nonetheless, this development made me fall "in love" with FreeBSD as it worked better for my needs than anything else. Fast forward to FreeBSD 9.2 and I still consider it to be the best distribution for the server and desktop environment support has been improving ever since.

Still, FreeBSD offered difficulties, as FreeBSD is tailored to the server environment, so getting set-up in a desktop environment can require a lot of command line work just to get all the services running properly. A friend of mine then introduced to me to PC-BSD, a desktop oriented operating system based on FreeBSD, which became my solution for some months. PC-BSD works very well and is great for beginners, but it is still a "heavy" system and you still have to use the command line to use ports. Their App Cafe is great for beginners who just want to have it running and do not have any knowledge about what is happening, but for advanced users it is less than ideal because it runs more slowly than straight FreeBSD. It also is getting to the point where it is difficult to run on older hardware. Another issue is that if you decide to use ports, you run the risk of App Cafe applications breaking, which happened to me on more than one occasion.

This year I also started to use the FreeBSD forums and meet a lot of very nice and helpful people who post there, which led me to meet Eric Turgeon. I continued to jump from FreeBSD to PC-BSD and vice versa until I heard Eric talk about GhostBSD. I refused to even take a look till Gnome 2.2 started to give me a lot of problems and FreeBSD does not port to Gnome 3, as the underlying architecture is not supported. Combined with Gnome 2.2 no longer being supported, more problems arose.

From there, I found that the Mate desktop environment was the project fork for Gnome 2 and it worked very well at first sight and remembered that GhostBSD already comes with Mate support. So, I downloaded GhostBSD 3.5-BETA1 and all the coming versions, testing it a lot. I was pleasantly surprised at how everything I needed just worked. In fact, GhostBSD worked so well that I got in contact with Eric and he invited me to become part of the project.

GhostBSD: The right fit for the desktop

If you want to use FreeBSD on the Desktop, GhostBSD is a very strong solution. I recommend it for beginners, normal and advanced users. The GhostBSD installer makes installation a breeze and it has his own application manager which works well with ports without any problem. Installing with packages is another option, but if I have the time, installing via ports is my preferred method as all proper dependencies are installed along with the applications.

GhostBSD also comes with a variety of windows managers to choose from. My preference being Mate as it offers a lot of functionality and runs well even on older equipment, but Gnome, XFCE, LXDE, and Openbox are good options as well. GhostBSD pre-configures the most common software choices users prefer for FreeBSD, fine-tuning for optimal performance. This allows users to avoid the process of extensive configuration, building and compiling their own FreeBSD system. GhostBSD is configured for low resource consumption and stability, while not limiting the user's customization options normally found in FreeBSD. Also, all of the tutorials, advice, and online content applicable to FreeBSD apply to our distribution.

While GhostBSD is still early in its development, it takes advantage of many of the features already found in FreeBSD. GhostBSD comes with video card and Wi-Fi support out of the box, which can be a big headache for inexperienced users. It also supports FreeBSD next generation package management system, Apache Open Office, Libre Office, LibreCAD, Eclipse/Anjuta development environments and much more. As the project matures and expands it will continue to add features while meeting its primary goals of security and useability in a lightweight installation.

Getting GhostBSD

To try out GhostBSD for yourself, you will first need to enter the GhostBSD web site at *http://ghostbsd.org* and go to the download section. From there, just choose the option you want from the provided list. There are options for i386 and 64bit, USB or disc images, and default desktop environments. After downloading to your preferred media, make sure you have your boot priorities set up correctly in BIOS and then start up your system. The graphic interface will then load and you will need to click on the icon to install GhostBSD.

The installation GUI will ask you to choose the partition to install, whether to use the GhostBSD boot loader (so when the system starts you can choose what system do you want to run), as well as other basic options like root and user password. After you finish the series of questions, installation begins and it's just a matter of time to finish, usually 10 or 20 minutes. When installation is over it will ask you to reboot. Make sure to remove your boot media when rebooting, and you have just installed GhostBSD.

Post-Installation

Once GhostBSD starts, just log in and you will see why GhostBSD is very powerful as a desktop. Load times are very fast, which is immediately noticeable when you click on an icon or on any application you want to run. Speaking of applications, popular common programs like Libre Office, Firefox, Brasero and others come pre-installed so you can get to work right away. GhostBSD also comes with a wi-fi manager that is very intuitive. Just open it and you will see all the connections inside your wireless range. Simply choose what you want to use and you will see all the options for it. While many of these options are also available in PC-BSD, you can test for yourself to see that GhostBSD is faster due to its design.

In addition, GhostBSD comes with a very helpful package manager. Just select it from the drop down menu, and then select update from the list and you can choose whatever you wish to install from the list and it will download and install it for you automatically. Also if you like to use ports like me, you can do it with no fear. You won't "break it", just download and update the port list and use it like you would do in a FreeBSD system. GhostBSD also provides community support on the Forum around the clock, since our team is from several different time zones.

To conclude, GhostBSD offers a secure, stable lightweight BSD installation with a full set of utilities pre-configured so you can hit the ground running. If you are a new or advanced user, GhostBSD takes away the hassle of configuration while providing all the powerful tools of a FreeBSD system. With its lightweight, full featured Desktop Environment options, GhostBSD offers you a powerful solution regardless of skill level or top of the line hardware.

ADRIAN J. PANUNZIO

How Secure Can Secure Shell (SSH) be?

(One Time Password aka OTP)

This article is the second part of the OpenSSH and demonstrates configurations as well as tricks that make using the protocol more secure.

What you will learn…

- How to configure OTP for your needs.
- A good base to make up something new and secure on your own.

What you should know…

- Unix/Linux commands and SHELL environments.
- The basics of TCP/IP.
- Basic configuration of SSH (1st part of the series)
- Understanding of security is necessary.

To begin, let's concentrate on the One Time Password (OTP). We are going to achieve our already secure SSH in conjunction with OTP for remote system connections. At first, in algorithmic meaning, OTP is a character string which should never repeat. However, "never" is a notion near infinity that never achieves it. Secondly, OTP has a discrete form of existing. The lifetime is finite and stands unchanged for seconds, minutes, possibly months or years.

Imagine you have a system which generates a new character string every minute, and let's name it as our OTP. The exemplary system uses 26 characters of an alphabetical array [a, b, …, z] (lower-case letters only) and 10 characters of digits [0, 1, …, 9]. OTP is 16 characters long and includes characters from the mentioned arrays. So, we can obtain from the example such character strings:

```
rwsyqhz45gtbuwhd
gbmmx5dlcytq60in
t27l5m86yqkslvb0
```

How many different character strings can we generate? What is the probability of guessing the correct character string? Let's go back to the math.

Count of the number of possible characters to use: 26+10=36 [a, b, …, z, 0, 1, …, 9]

Length of the character string (OTP): 16. Now, step-by-step:

1) The first character of the string can be randomly selected as 1 from 36, hence we have 36 options of that.
2) The second character of the string can be randomly selected as 1 from 36 as well, and hence we still have 36 options for this character.
3) The third character of the string can be randomly selected as 1 from 36 and have 36 options as well.

…

16)The last character of the string can be randomly selected as 1 from 36 and we have 36 options.

We assume that the characters can be the same and, in this particular case, all 16 characters can be the same. A combination of all characters in our 16 long string is equal (step-by-step).

1) We have a one character string: 36 different character strings.
2) We have a two character string: 36 multiply 36 is equal to 1296 (36*36=1296) different character strings.
3) We have a three character string: 36*36*36= 46656 (36^3=46656) different character strings.

…

16)We have a sixteen character string: 36^{16}= 79586 61109946400884391936 different character strings.

Is the number huge? Let's check.

We will assume that we change our OTP once every minute and the OTP "never" repeats. There are 1440 minutes in one day. There are 365.25 days in one year.

For how many days shall we have different character strings?

7958661109946400884391936 divided by 1440 is equal to 5526847993018333947494.4 days.

For how many years shall we have different character strings?

5526847993018333947494.4 divided by 365.25 is equal to around 15131685128044719911 years.

Around 15 quintillion years (in short scale) for IT guys, it's the same as $15*10^{18}$ years (15 ExaYears). If you are interested in learning all the scales, (short and long), look at the website *en.wikipedia.org/wiki/Long_and_short_scales*.

To compare that number with something similar, know that the lifetime of the proton (predicted) is equal to $3*10^{40}$ seconds, the age of the universe is equal to $5*10^{17}$ seconds, and our number of different OTPs is equal to around $7.95*10^{24}$. The probability of guessing the OTP is 1 to $7.95*10^{24}$.

The machine, (computer, especially CPU), can work for one minute to find out the OTP. The probability of guessing the OTP during one minute by the fastest computer in the world (*www.top500.org*) with a performance of 33.86 petaflops per second ($33.86*10^{15}$ flops/s) is around $(33.86*10^{15})*60 / 7.95*10^{24} = 2{,}55*10^{-6}$ (0,000255%). Note: it's a very simple comparison and there are many searching algorithms which could be used to speed up finding out the OTP, but the probability of the worst case scenario shows the scale.

We can argue that our 16 long One Time Password is secure enough. If we try to use not a 36 character array, but the ASCII table of characters, we shall have 128^{16} OTPs. Try to compute this and compare with the known universe mass equal to $1*10^{53}$!

Let's go back to our SSH. We will explore a few methods of generating and using an OTP but first, we shall get familiar with passwords on Unix systems.

Both FreeBSD and OpenBSD systems keep passwords in the `/etc/master.passwd` file and the user password is encrypted by one of the algorithms set in the `/etc/login.conf` (find string `:passwd_format=` in FreeBSD and `:localcipher=` in OpenBSD).

There are many hash functions that can be used to calculate the encrypted password: DES, Blowfish, MD5, SHA256, SHA512 etc. By default FreeBSD uses SHA512 and OpenBSD uses Blowfish with 6 iterations.

It's good to know more about the content of the master.passwd file which stores the encrypted user's passwords.

Look at the command below:

```
# cat /etc/master.passwd | grep John
John:$1$4yQeiBqO$AZOv/r0Q4DcxkF5KvcKN8/:1001:0:admin:0:0:J
    ohn Buzz:/home/John:/bin/sh
```

Special data is separated by the $ sign and the hashed password `$1$4yQeiBqO$AZOv/r0Q4DcxkF5KvcKN8/` is described below:

`1` means MD5 algorithm.
`4yQeiBqO` is a salt used to make the password more difficult to find out. Salt is generated by the first two characters of the encrypted password.
`AZOv/r0Q4DcxkF5KvcKN8/` is the encrypted password.

As previously stated, FreeBSD uses SHA512 by default, (which would display the number 6 behind the first `$ sign`), but one can change it to MD5 by adding the following lines to `/etc/login.conf`.

```
admin:\
          :passwd_format=md5:
```

To apply the above changes, run the following command.

```
# cap_mkdb /etc/login.conf
```

The same command works for OpenBSD as well. Note: the MD5 hash function is worse than SHA512. Worse means that it takes less computing to break the hash. I only showed as an example, how generating a new password should work and how to change the hash function. If your system still uses DES, MD5 or other weak functions, change it.

You could come up with the idea to just replace the string in the master.passwd file, but it's not as easy as coming up with the idea. Let's use third party applications to generate our OTP.

For security and information reasons, try to tinker and change some of the following values for a particular user or a group in the login.conf file. Are some of them combined with SSH? Yes: "welcome" for sure. The rest of the options should be clear. Default values from my OpenBSD and FreeBSD respectively. For more information about the login.conf file, look at the command man login.conf.

```
default:\
          :path=/usr/bin /bin /usr/sbin /sbin /usr/X11R6/bin
    /usr/local/bin /usr/local/sbin:\
          :umask=022:\
          :datasize-max=512M:\
          :datasize-cur=512M:\
          :maxproc-max=256:\
          :maxproc-cur=128:\
          :openfiles-cur=512:\
          :stacksize-cur=4M:\
          :localcipher=blowfish,6:\
          :ypcipher=old:\
          :tc=auth-defaults:\
          :tc=auth-ftp-defaults:

default:\
          :passwd_format=sha512:\
          :copyright=/etc/COPYRIGHT:\
          :welcome=/etc/motd:\
          :setenv=MAIL=/var/mail/$,BLOCKSIZE=K:\
          :path=/sbin /bin /usr/sbin /usr/bin /usr/games /
    usr/local/sbin /usr/local/bin ~/bin:\
          :nologin=/var/run/nologin:\
          :cputime=unlimited:\
          :datasize=unlimited:\
          :stacksize=unlimited:\
          :memorylocked=64K:\
          :memoryuse=unlimited:\
          :filesize=unlimited:\
          :coredumpsize=unlimited:\
          :openfiles=unlimited:\
          :maxproc=unlimited:\
          :sbsize=unlimited:\
          :vmemoryuse=unlimited:\
          :swapuse=unlimited:\
          :pseudoterminals=unlimited:\
          :priority=0:\
          :ignoretime@:\
          :umask=022:
```

FreeBSD and OpenBSD use different applications to generate the OTP for us. OpenBSD uses S/Key system and FreeBSD uses OPIE. Let's start for FreeBSD.

FreeBSD OTP by OPIE (One-Time Passwords in Everything)

There is a seed that consists of two letters, five digits and an iteration count. The OTP is created by concatenating a secret password with the seed and then applying the hash function MD5 as many times as the iteration count states. Then the OPIE turns the result into six words as our OTP.

Let's assume that we are logged in to our system via SSH. Here are the step-by-step instructions to run OTP.

Initialization (please be logged on as a standard user not privileged):

```
# opiepasswd -c
```

And output (system requests for the pass phrase for user John):

```
Adding John:
Only use this method from the console; NEVER from remote.
   If you are using
telnet, xterm, or a dial-in, type ^C now or exit with no
   password.
Then run opiepasswd without the -c parameter.
Using MD5 to compute responses.
Enter new secret pass phrase:
Again new secret pass phrase:

ID John OTP key is 499 mo3726
TACK LOP AN ADEN GIFT BEND

499 – sequence number
mo3726 – seed
```

Listing 1. *SSH successful connection screenshot*

```
Using username "John".

                         Access Restricted Equipment
                   All Activities are Monitored and Logged
                         Unauthorized Use Prohibited

    By Accessing, You Are Agree Your Activities to be Monitored and Logged

Authenticating with public key "imported-openssh-key"
Passphrase for key "imported-openssh-key":
Further authentication required
Using keyboard-interactive authentication.
otp-md5 498 mo3726 ext
Password:
Last login: Mon Nov 25 18:13:02 2013 from 192.168.0.18
FreeBSD 9.2-RELEASE (GENERIC) #0 r255898: Thu Sep 26 22:50:31 UTC 2013

                                 MATRIX
----------------------------------------------------------------------------------
| 984653 | 760500 | 786864 | 727064 | 374556 | 263648 | 777138 | 345072 | 428465 |
| 859924 | 468676 | 277695 | 743340 | 782600 | 955084 | 537264 | 847652 | 991796 |
| 585728 | 637052 | 580840 | 340080 | 471782 | 816764 | 780728 | 413452 | 135408 |
| 255580 | 676208 | 385216 | 280644 | 630136 | 772460 | 787592 | 856622 | 671164 |
| 391144 | 976716 | 106548 | 460668 | 142948 | 559468 | 213668 | 439140 | 488332 |
| 296764 | 705276 | 189176 | 420836 | 657904 | 404196 | 276900 | 956540 | 532012 |
| 167180 | 734240 | 395928 | 597428 | 385632 | 930436 | 609388 | 609700 | 401492 |
| 806624 | 694304 | 277345 | 644072 | 554320 | 711256 | 705692 | 233896 | 380380 |
| 144508 | 657488 | 994344 | 162396 | 598468 | 502344 | 525037 | 299676 | 448136 |
----------------------------------------------------------------------------------

Attempts left: 3.
Unlock key:
Terminal unlocked!
$

Using username "John".

                         Access Restricted Equipment
                   All Activities are Monitored and Logged
                         Unauthorized Use Prohibited

    By Accessing, You Are Agree Your Activities to be Monitored and Logged
```

At this time, if user John tries to log in via SSH, he is asked to type the OTP. Where is the password?

Password is generated on the other machine using the following command:

```
$ opiekey 498 mo3726 ext
```

Using the MD5 algorithm to compute response.

Reminder: Don't use opiekey from telnet or dial-in sessions. Enter secret pass phrase:

```
WAS KURD LOG MONA BONE DRUG
```

Copy `WAS KURD LOG MONA BONE DRUG` into the terminal where you're trying to log in and asking for a password. The Listing 1 depicts the result.

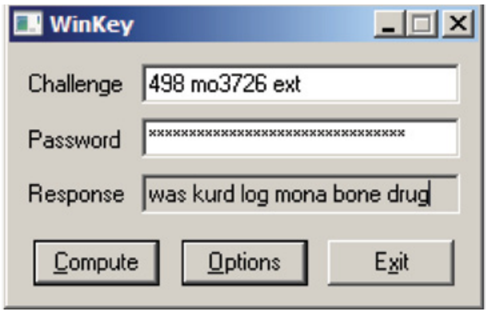
If you didn't read the first article, please be informed that MATRIX, Attempts, left and Unlock key texts are my own application prompts. You can try it by downloading from *www.iptrace.pl* (go to Download and click on a Download Locker button). The application is free of charge on the BSD Licence. Please send any suggestions and bugs found at the Locker via e-mail to *locker@iptrace.pl*. If you want to generate more than one OTP, run the following command. Option `-n` and then a value to indicate the number of OTPs.

```
$ opiekey -n 5 498 mo3726 ext
```

Using the MD5 algorithm to compute response.

Reminder: Don't use opiekey from telnet or dial-in sessions. Enter secret pass phrase:

```
494: LAY REAL RASH JUJU LANG LINE
495: SAIL DOCK TILE MIRE SOY NULL
496: YAM WEAR ROAM FIST TWIN SUE
497: TRAM CANT FOLK AFRO OVA BAND
498: WAS KURD LOG MONA BONE DRUG
```



Challenge 498 mo3726 ext
Password ********************************
Response was kurd log mona bone drug

Figure 1. *MS Windows WinKey One Time Password generator*

Please bear in mind that it's not a good solution to generate a password on the first system to log into a second one. If you want to have all SSH terminals (systems) blocked by OTP use an MS Windows application to generate an OTP. See the following screenshot of that application. You can download WinKey from *ftp://ftp.irisa.fr/pub/OTP/*.

Note: the OTP generated by OPIE doesn't change the real UNIX passwords in `master.passwd` file. To disable using OPIE run the following command.

```
$ opiepasswd -d John
```

To allow logging in for users from specified IP addresses or networks via UNIX password and bypass OPIE, change the settings in the `/etc/opieaccess` file. But you can still use OTP if needed. So you have two ways to get the system.

OpenBSD OTP by S/Key (One-Time Passwords in Everything)

The S/Key uses a secret pass phrase with challenge. Conceptually, the workings of S/Key are similar to OPIE.

Let's assume that we are logged in to our system via SSH. Here are the step-by-step instructions to run the OTP generator.

Initialization (please be logged on as a privileged user, root) to create the `/etc/skey` directory:

```
# skeyinit -E
```

Re-login as a standard user, for my example the user is John, then run the following command.

```
# skeyinit
```

And output (system requests for the pass phrase for user John):

```
Reminder - Only use this method if you are directly
   connected
   or have an encrypted channel.  If you are using telnet,
   hit return now and use skeyinit -s.
Password:
[Adding John with md5]
Enter new secret passphrase:
Again secret passphrase:

ID John skey is otp-md5 100 utm167228
Next login password: SEAL TEEN FROG HAWK WADE RID
```

Yes, you're right. It's almost the same as for OPIE. So, it's easy to go through the rest of the tour.

Listing 2. *SSH successful connection screenshot*

```
Authenticating with public key "imported-openssh-key"
Passphrase for key "imported-openssh-key":
Further authentication required
Using keyboard-interactive authentication.
otp-md5 92 utm167228
S/Key Password:
Last login: Tue Nov 26 00:42:41 2013 from 192.168.0.18
OpenBSD 5.3 (GENERIC) #50: Tue Mar 12 18:35:23 MDT 2013

            #     #
            #     #  #    #     #    ######     #    ######  #####
            #     #  ##   #     #    #          #    #       #    #
            #     #  # #  #     #    #####      #    #####   #    #
            #     #  #  # #     #    #          #    #       #    #
            #     #  #   ##     #    #          #    #       #    #
             #####   #    #     #    #          #    ######  #####

            #######
               #     #    #  #####   ######    ##     #####   ####
               #     #    #  #    #  #        #  #      #    #
               #     ######  #    #  #####   #    #     #     ####
               #     #    #  #####   #       ######     #         #
               #     #    #  #   #   #       #    #     #    #    #
               #     #    #  #    #  ######  #    #     #     ####

#     #
##   ##    ##    #    #    ##     ####  ######  #    #  ######  #    #   #####
# # # #   #  #   ##   #   #  #   #    #  #       ##  ##  #       ##   #     #
#  #  #  #    #  # #  #  #    #  #       #####   # ## #  #####   # #  #     #
#     #  ######  #  # #  ######  #  ###  #       #    #  #       #  # #     #
#     #  #    #  #   ##  #    #  #    #  #       #    #  #       #   ##     #
#     #  #    #  #    #  #    #   ####   ######  #    #  ######  #    #     #

                            ###             ###    #####
                           #   #           #   #  #     #
                          #     #         #     #       #
                          #     #         #     #  #####
                   #    # #     #         #     #       #
                    #  #   #   #    ###    #   #  #     #
                     ##     ###     ###     ###    #####


                                 MATRIX
----------------------------------------------------------------------------------
| 518760 | 398556 | 818784 | 893601 | 999198 | 797949 | 274554 | 509382 | 832707 |
| 169866 | 754128 | 333099 | 419526 | 898398 | 749124 | 276210 | 279414 | 404874 |
| 532629 | 260586 | 269505 | 190323 | 775116 | 958446 | 665856 | 167031 | 465210 |
| 993285 | 624825 | 414144 | 126333 | 832734 | 509823 | 637713 | 596691 | 896436 |
| 713673 | 991665 | 797661 | 138420 | 397791 | 719703 | 518040 | 630180 | 242181 |
| 676926 | 435033 | 266652 | 563229 | 785772 | 335277 | 456669 | 490824 | 823554 |
| 275486 | 365419 | 473229 | 914166 | 974439 | 881991 | 200709 | 564075 | 264825 |
| 838161 | 378396 | 257517 | 821394 | 889425 | 261360 | 305919 | 727308 | 855540 |
| 732159 | 884286 | 520020 | 804681 | 918837 | 757737 | 203904 | 903870 | 681444 |
----------------------------------------------------------------------------------

Attempts left: 3.
Unlock key:
Terminal unlocked!

$ su -a passwd -l
Password:
```

```
# skey 100 utm113739
Reminder - Do not use this program while logged in via
    telnet.
Enter secret passphrase:
SARA CHIN WATT KNEW CUB SCOT
```

Once again, if you want to generate more than one OTP, run the following command. Option `-n` and then a value that indicates the number of OTPs.

```
utm1:~> skey -n 5 100 utm113739
Reminder - Do not use this program while logged in via
    telnet.
Enter secret passphrase:
96: DOES BLAT TILT NOLL NARY HUT
97: WARN TWIG FREE TRAY SIGH AIDE
98: LENT BURN GEL GOES CHAD LOOT
99: SHOW AWE TINA LIED WATT WANT
100: SARA CHIN WATT KNEW CUB SCOT
```

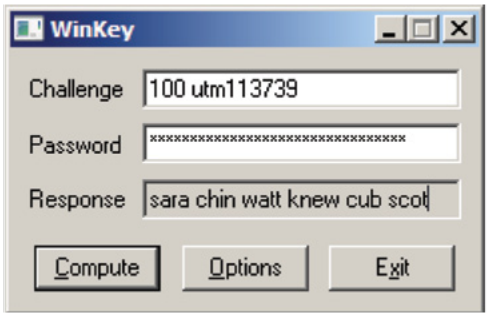
Again, we use WinKey to generate OTP (Figure 2).

The last one change in OpenBSD is to replace one row in the login.conf file. See what data should be correct.

```
auth-defaults:auth=skey,passwd:
```

For security reasons change the second line as well.

```
auth-ftp-defaults:auth-ftp=skey,passwd:
```



Challenge: 100 utm113739
Password: **********************************
Response: sara chin watt knew cub scot

Figure 2. *MS Windows WinKey One Time Password generator*

To apply the above changes run the following command.

```
# cap_mkdb /etc/login.conf
```

If you're not going to use password authorization in the near future, delete the password value from the above to enforce using the OTP only. It does not work for root, who always can get on the system using a standard Unix password.

Note: we don't generate the OTP for a privileged user, root, due to maintenance, to not make the authentication track more complicated, and for ease of using an account from the console. You have to know that OpenBSD behaves differently than FreeBSD during logging using the `su` command. Even though we have not created the OTP for root, OpenBSD asks for it. To prevent this use the following command when you log in from a standard user.

```
# su -a passwd
```

Let's look for the logging successful process (Listing 2).

Remember, if a counter is going to 0 (zero), it's important to reinitialize the counter again. Use the following commands for OpenBSD and FreeBSD respectively, otherwise you won't be able to log in.

```
# skeyinit
# opiepasswd -c
```

Conclusions

Using One Time Passwords (OTPs) is a very good approach to improving system authorization security. In conjunction with public/private keys, Unix passwords and some OpenSSH defences, OTP ensures great security without too much of a decrease in functionality.

In the next series you will find out more about:

- VPN tunnelling – creating Virtual Private Networks using OpenSSH
- SFTP – known as SSH File Transfer Protocol to opposite of a standard FTP

References (in order of relevance)

man opiepasswd (FreeBSD)
man opiekey (FreeBSD)
man opieaccess (FreeBSD)
man skeyinit (OpenBSD)
man skey (OpenBSD)
man master.passwd
www.openssh.org; www.openbsd.org; www.freebsd.org

ARKADIUSZ MAJEWSKI, BENG

Arkadiusz Majewski comes from Poland. He has 15 years' experience with ICT technologies, including 15 years of IT networks, 10 years of BSD systems and MS Windows Server solutions. He has also 5 years' experience with programming languages and Telco solutions. He's interested in security on all business and ICT levels. In his free time he reads ICT books, and deepens his knowledge about science (math, physics, chemistry). His hobby is cycling and motorization. He's a graduate of Warsaw Information Technology under the auspices of the Polish Academy of Sciences. Feel free to contact the author via e-mail bsd.magazine@iptrace.pl.

UPDATE NOW WITH STIG AUDITING

"IN SOME CASES **nipper studio** HAS VIRTUALLY **REMOVED** the **NEED FOR** a **MANUAL AUDIT**"

CISCO SYSTEMS INC.

Titania's award winning Nipper Studio configuration auditing tool is helping security consultants and end-user organizations worldwide improve their network security. Its reports are more detailed than those typically produced by scanners, enabling you to maintain a higher level of vulnerability analysis in the intervals between penetration tests.

Now used in over 45 countries, Nipper Studio provides a thorough, fast & cost effective way to securely audit over 100 different types of network device. The NSA, FBI, DoD & U.S. Treasury already use it, so why not try it for free at www.titania.com

2012 Computing Security Awards WINNER Enterprise Security Solution of the Year

Security Products GOVIES 2013 Government Security Awards

2012 Computing Security Awards WINNER Network Security Solution of the Year

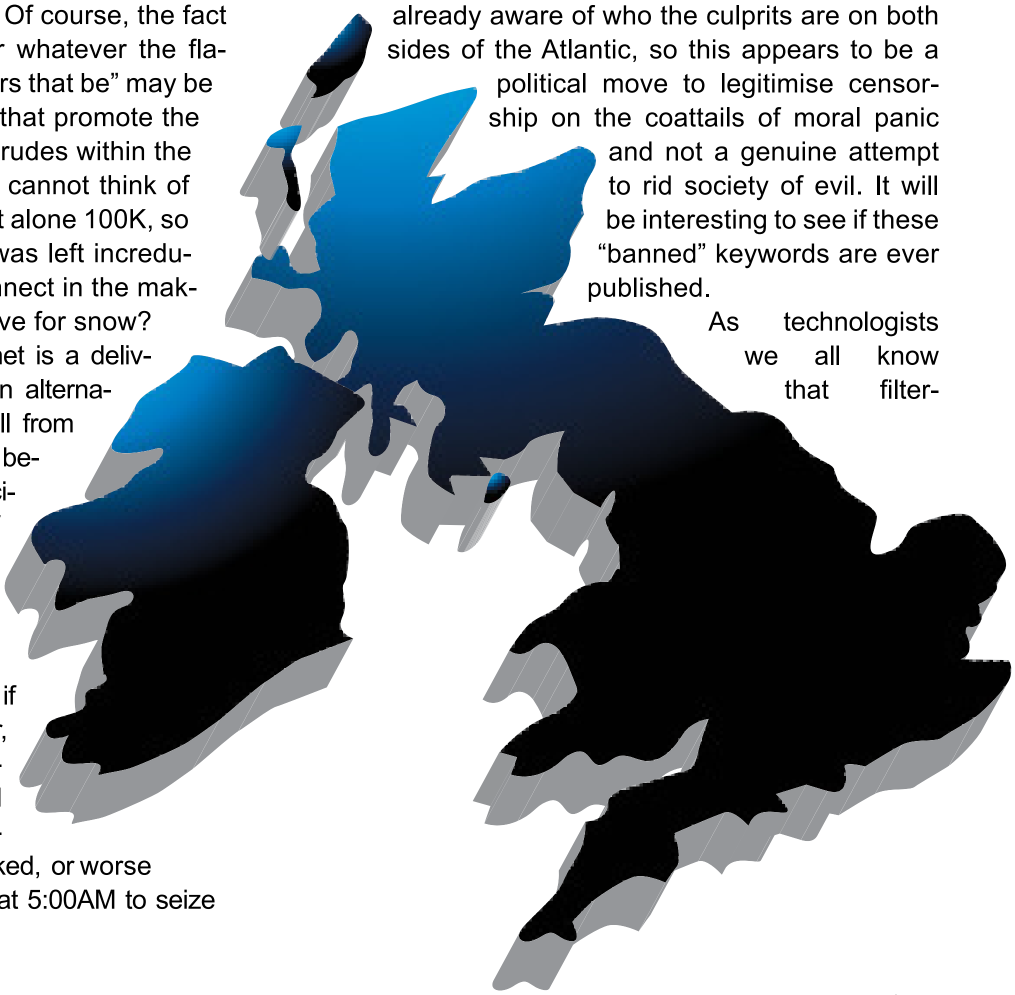
www.titania.com

OPINION: With the UK government in collusion with the major search engines to censor 100,000 search terms to prevent child abuse, is the UK joining the ranks of the technological fascists?

David Cameron, while no fool, by his authority as Prime Minister of the United Kingdom, has backed the censorship of 100K search terms alluding to child abuse in collusion with Google, Bing, and no doubt other search providers accessible in the UK. The exact extent of the legal framework is yet to be formalised, but it is clear that the UK government is moving towards a more proactive stance of censorship in a populist move to assuage the "something must be done to protect us from the Internet" lobby. Of course, the fact that political affiliations, terrorism, or whatever the flavour of the day that offends "the powers that be" may be added to this list has escaped those that promote the nose of this particular camel that protrudes within the tent of content delivery. Personally, I cannot think of 100 terms that relate to child abuse let alone 100K, so my inner skeptic, not unexpectedly, was left incredulous. A classic case of political disconnect in the making. How many words do Eskimos have for snow?

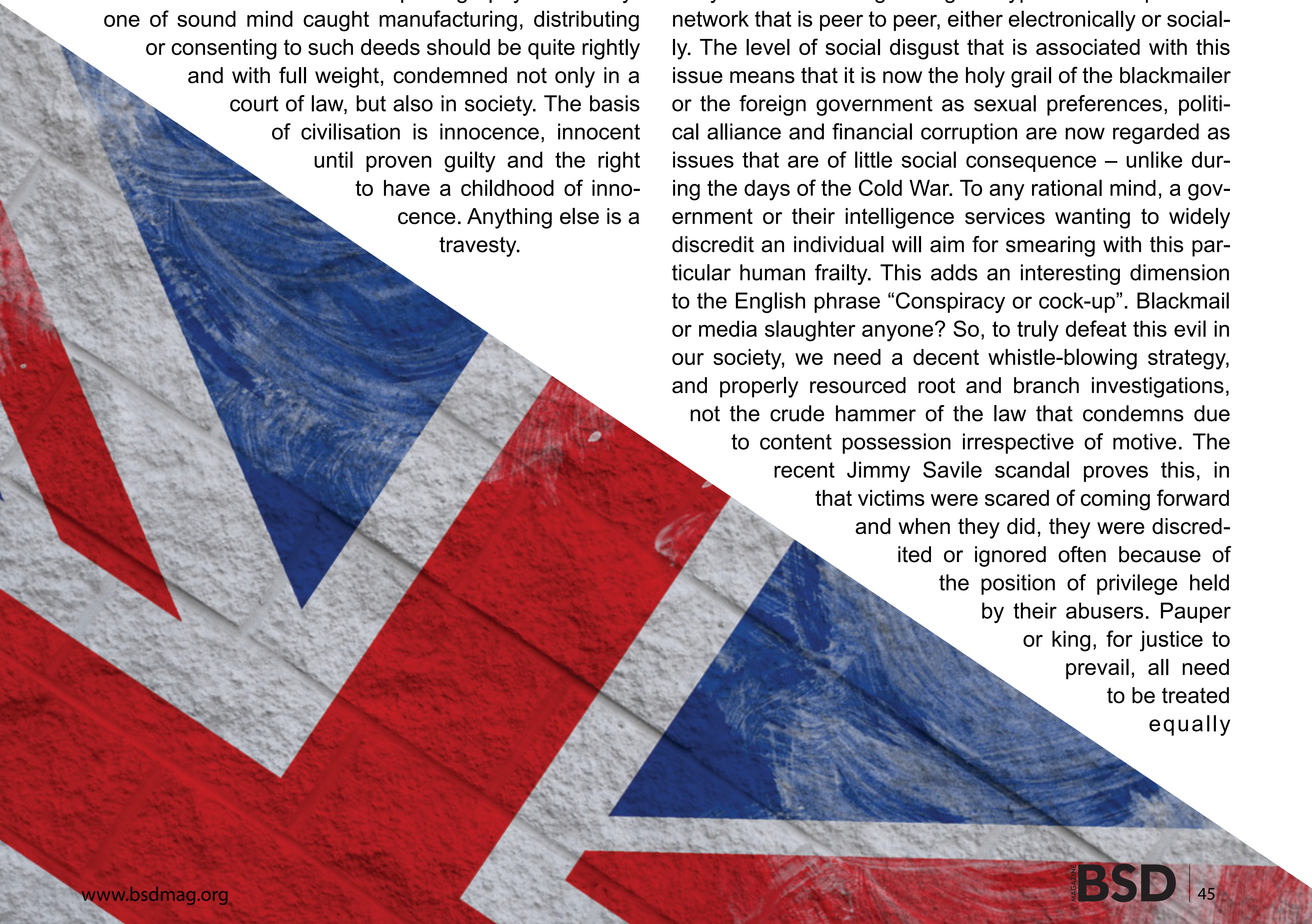
Contrary to popular belief, the Internet is a delivery system, not some monster with an alternative agenda to deprave and corrupt all from conceived embryos to the elderly and beyond the grave. It is a reflection of society. On the surface, triggering an alert if someone was to type "kiddie porn" into Google, seems a good way to deal with the totally abhorrent desire of an individual to have sexual relations with prepubescent children. What happens if you are a genuine journalist, researcher, concerned parent or a medical professional? Your browser gets an alert and your IP address is committed to a database. Then what? Questions are asked, or worse case, a visit by your local police force at 5:00AM to seize all Internet enabled devices, recordable media and a forensic investigation of every detail of your life and moral censure? The Internet is transient – a page can appear and disappear within minutes, or in the case of the current Conservative governments' previous election promises – a few years. Thanks for nothing, Google. What is still unclear is how much of this data will be passed to other intelligence services or bodies via the NSA and GCHQ.

Let's not be under any illusion here, the watchers are already aware of who the culprits are on both sides of the Atlantic, so this appears to be a political move to legitimise censorship on the coattails of moral panic and not a genuine attempt to rid society of evil. It will be interesting to see if these "banned" keywords are ever published.



As technologists we all know that filtering search terms and attempting to categorise them intelligently is a pretty pointless exercise unless you throw massive human resources at it. During the miners' strike in the UK during the 80s, the eavesdropping system monitoring UK telephone conversations was overloaded due to the sheer weight of relevant data. I recently installed a corporate wide messaging system on our Intranet, and as a precaution to assuage the naysayers, added a swear filter knowing full well that it was a token gesture. If people want to do bad things, they will find a way to do them. This is IT help-desk 101. The fallacy that technology can be a moral guardian is rife with miscarriages of justice. Just ask any motorist who has been captured speeding by a badly aimed or calibrated speed gun or, indeed, a customer on the wrong end of a customer services "script". Technology is digital, black and white, whereas real life is analogue, a spectrum of colour. Here lies the perpetual paradox and argument between the spirit and the letter of the law. Unfortunately, history has proven that venal individuals can capitalise on this argument, be they defendants, prosecutors or, notably, governments.

So let's cut to the chase. Child pornography is evil. Anyone of sound mind caught manufacturing, distributing or consenting to such deeds should be quite rightly and with full weight, condemned not only in a court of law, but also in society. The basis of civilisation is innocence, innocent until proven guilty and the right to have a childhood of innocence. Anything else is a travesty.



Unfortunately, the law once again blindly overreaches in this regard, as possession in the UK of the worst type of pornography is a strict liability offence (i.e. you got it, you are guilty). While no cases to my knowledge have reached the courts here in the UK, the law is cut and dried – if you have "bad content" on your servers, you are liable. End of story. No *mens rea* (state of mind) appeal is allowed under strict liability cases. So as a system administrator in the UK, if I find objectionable 3rd party material on my server I run the risk of prosecution if I attempt to hand this material over to the authorities. So what should I do? Delete it and say nothing? In theory, no prosecuting authority would be so aggressive as to pursue such a case with a co-operating individual but in this age of febrile condemnation of the mass media and legalism, who knows? If somebody wanted to prove a point, all they need do is dump some images on a competitor's or political opponent's hard disk and make a few phone calls. The rules and ethics that work in the real physical world (e.g. possession of drugs) does not work with electronic data.

In reality, the neighbourhood paedophile is protected. They are either using strong encryption or are part of a network that is peer to peer, either electronically or socially. The level of social disgust that is associated with this issue means that it is now the holy grail of the blackmailer or the foreign government as sexual preferences, political alliance and financial corruption are now regarded as issues that are of little social consequence – unlike during the days of the Cold War. To any rational mind, a government or their intelligence services wanting to widely discredit an individual will aim for smearing with this particular human frailty. This adds an interesting dimension to the English phrase "Conspiracy or cock-up". Blackmail or media slaughter anyone? So, to truly defeat this evil in our society, we need a decent whistle-blowing strategy, and properly resourced root and branch investigations, not the crude hammer of the law that condemns due to content possession irrespective of motive. The recent Jimmy Savile scandal proves this, in that victims were scared of coming forward and when they did, they were discredited or ignored often because of the position of privilege held by their abusers. Pauper or king, for justice to prevail, all need to be treated equally under the law. Sadly, this is not the case. God help an innocent ISP or a victim under the current legislation.

David Cameron's febrile attempt at cleaning up the Internet proves beyond all doubt that he doesn't understand the issues. Over 90% of child abuse victims know their abuser socially. Granted, the Internet is a medium that allows people to build relationships, but to categorise an individual as deviant by what request they submit to a search engine is not only an abuse of process, but an abuse of power. And that doesn't take into account the malware a reasonably skilled IT engineer could build to generate a spoof of an individual's request. This move plays right into the hands of the spammers and the criminal underworld, allowing them to blackmail ordinary citizens with false accusations. "You have been looking at illegal content. Send us your credit card details and £250 or we contact the authorities". No paedophile is going to be searching for the type of content they desire using a search engine – it is more likely to be distributed via peer-to-peer or stored on a server within the Tor network. The truly paranoid would send it via snail mail on an encrypted USB stick or CDROM. So this cure will create more problems than it solves.

So what can we do about this evil as a community? First of all, we all need to be aware of and identify all the different types of low-life that are out there – fraudsters, sock-puppets, trolls, spammers, bandwidth abusers and copyright infringers, *et al* irrespective of whether we are IT professionals or end users. Birds of a feather flock together. I am not generally talking about individuals here, as we are are probably all guilty at some point of committing some of these actions to a lesser degree. Who hasn't filled in a web-form with false details or used the corporate network to download an MP3 or two? I am talking about the communities that make a lifestyle, political or commercial choice to do such things en-mass on a regular basis causing disruption and distress to all.

We need a mechanism to quickly electronically disable and deal with these communities in law. If you get 500 phishing emails a day, that is 500 counts of attempted fraud, but will law enforcement take it seriously? Due to the distributed nature of networks, while the malware causing the problem may be on 500 individuals' PC's, it is not necessarily true that they are guilty of anything other than bad security hygiene. It is the authors and bot-masters who are guilty. We need a segregation of legal adult content into a XXX domain that is easily blocked by parental controls, backed by legislation that pursues the owner of the domain (e.g. the content owner) for breach. The province of the purchaser can then easily be proved in a court of law, absolving the ISP of responsibility. After all, like an estate agent or realtor they are only selling space, they are not responsible for the acts that take place inside the property. Of course, if the ISP does discover illegal activity, they have a duty to report it. Still, as mentioned earlier, in the UK at least it is not that easy. The same idea could apply for global financial transactions etc., but of course certain vested interests want to have their cake and eat it, in that they want global freedom without necessarily any accountability or responsibility.

So a global Internet wide agreement is probably never going to happen.

Another approach is on a country by country basis. Once again, this has its dangers. I don't want some policy maker deciding if I can visit *www.ihatemygovernment.org* (Yes. It exists). China and Google firewalls anyone? Anyhow, any experienced IT user can proxy or tunnel their way around it.

No, the Internet, like rain, sunshine and death is available to everyone, including paedophiles. The maxim "I disapprove of what you say, but I will defend to the death your right to say it" needs to be revisited and reconsidered as in 2013 we don't just have words but images and video available to all as well. While freedom of expression is vitally important we equally need social, moral and legal responsibility, from the tramp to the millionaire. We live in a wonderful age, where barriers are collapsing and we can connect and understand more than the shallow political rhetoric that has dominated the last 2000+ years. What matters most is what people and society values – in real life and online. Until we get some cohesive action and the issue of Internet crime is taken seriously just as it would be on the street, WWW will continue to stand for Wild Wild West.

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.

Is your MISSION-CRITICAL security strong enough to stop a SKILLED ATTACKER?

Don't guess
Don't believe
Don't hope
KNOW!

acros

An ACROS Penetration Test is conducted exactly like a real attack by a skilled, motivated adversary – only without the damage. We will find the weakest links in your security and use all our knowledge, skills and capabilities to try to achieve exactly what your security measures and policies are there to prevent. If it sounds difficult, we're interested.

Experience the ultimate test of your security.
(After all, the only alternative is to wait for an actual attack.)

ACROS Security – http://www.acrossecurity.com – security@acrossecurity.com

Headquarters:
San Jose, CA

855.GREP.4.IX | Contact Us

99% Compatibility

online now...

IXSYSTEMS AND YOU ARE
THE PERFECT MATCH

SHARED INTERESTS

- ✓ Enterprise Storage Solutions
- ✓ Personalized Customer Service
- ✓ 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