MAGAZINE

BSD

FOR NOVICE AND ADVANCED USERS

Beyond BIOS

THE EXTENDED FIRMWARE INTERFACE

DEBUGGING AND TROUBLESHOOTING

DEPLOYING AN OFFICE SERVER IN FREEBSD, WITH FILE SHARING AND E-MAIL

RETURN ORIENTED PROGRAMMING

VOL.8 NO.08
ISSUE 08/2014(61)
1898-9144

FREENAS MINI
STORAGE APPLIANCE

IT *SAVES* YOUR LIFE.

HOW IMPORTANT IS YOUR DATA?

Years of family photos. Your entire music and movie collection. Office documents you've put hours of work into. Backups for every computer you own. We ask again, *how important is your data?*

NOW IMAGINE LOSING IT ALL

Losing one bit - that's all it takes. One single bit, and your file is gone.

The worst part? **You won't know until you absolutely need that file again.**

Example of one-bit corruption

THE SOLUTION

The FreeNAS Mini has emerged as the clear choice to save your digital life. **No other NAS in its class offers ECC (error correcting code) memory and ZFS bitrot protection to ensure data always reaches disk without corruption and *never degrades over time*.**

No other NAS combines the inherent data integrity and security of the ZFS filesystem with fast on-disk encryption. No other NAS provides comparable power and flexibility. The FreeNAS Mini is, hands-down, the best home and small office storage appliance you can buy on the market. **When it comes to saving your important data, there simply is no other solution.**

The Mini boasts these state-of-the-art features:

- 8-core 2.4GHz Intel® Atom™ processor
- Up to 16TB of storage capacity
- 16GB of ECC memory (with the option to upgrade to 32GB)
- 2 x 1 Gigabit network controllers
- Remote management port (IPMI)
- Tool-less design; hot swappable drive trays
- FreeNAS installed and configured

iXsystems®

http://www.iXsystems.com/mini

intel® inside™ ATOM™

FREENAS CERTIFIED STORAGE

With over six million downloads, FreeNAS is undisputedly *the* most popular storage operating system in the world.

Sure, you could build your own FreeNAS system: research every hardware option, order all the parts, wait for everything to ship and arrive, vent at customer service because it *hasn't*, and finally build it yourself while hoping everything fits - only to install the software and discover that the system you spent *days* agonizing over **isn't even compatible**. Or...

MAKE IT EASY ON YOURSELF

As the sponsors and lead developers of the FreeNAS project, iXsystems has combined over 20 years of hardware experience with our FreeNAS expertise to bring you FreeNAS Certified Storage. **We make it easy to enjoy all the benefits of FreeNAS without the headache of building, setting up, configuring, and supporting it yourself.** As one of the leaders in the storage industry, you know that you're getting the best combination of hardware designed for optimal performance with FreeNAS.

Every FreeNAS server we ship is...

- » Custom built and optimized for your use case
- » Installed, configured, tested, and guaranteed to work out of the box
- » Supported by the Silicon Valley team that designed and built it
- » Backed by a 3 years parts and labor limited warranty

As one of the leaders in the storage industry, you know that you're getting the best combination of hardware designed for optimal performance with FreeNAS. **Contact us today for a FREE Risk Elimination Consultation with one of our FreeNAS experts.** Remember, every purchase directly supports the FreeNAS project so we can continue adding features and improvements to the software for years to come. **And really - why would you buy a FreeNAS server from *anyone* else?**

FreeNAS 1U

- Intel® Xeon® Processor E3-1200v2 Family
- Up to 16TB of storage capacity
- 16GB ECC memory (upgradable to 32GB)
- 2 x 10/100/1000 Gigabit Ethernet controllers
- Redundant power supply

FreeNAS 2U

- 2x Intel® Xeon® Processors E5-2600v2 Family
- Up to 48TB of storage capacity
- 32GB ECC memory (upgradable to 128GB)
- 4 x 1GbE Network interface (Onboard) - (Upgradable to 2 x 10 Gigabit Interface)
- Redundant Power Supply

http://www.iXsystems.com/storage/freenas-certified-storage/

Dear Readers,

You are going to read the "Beyond BIOS" issue from BSD magazine. You will learn how to prepare to install an EFI environment on an Intel-based, how to perform the installation and how to manage the computer once it's up and running. What is more, our experts will teach you with moderate experience in any Unix-like system to install and deploy a quality office server with common applications and services. Finally, you may find interest in the "Debugging and Troubleshooting: Fun, Profit and Go Home Earlier" tutorial provided by Carlos Antonio Neira Bustos. Carlos is going to represent a real life situation where debugging skills will save us time, headaches and possibly to find a solution using a minimal amount of effort.

I would like to express my gratitude to our experts who contributed to this publication and invite others to cooperate with our magazine.

The next issue of BSD Magazine will be published in 4 weeks. If you are interested in learning more about the future content or you would like to get in touch with our team, please feel free to send your messages to ewa.d@bsdmag.org. I will be more than pleased to talk and answer all your questions.

Hope you enjoy the issue.

Ewa Dudzic
and 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, Toki Winter, Wesley Mouedine Assaby, Rob Somerville

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

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.

FreeNAS

in an Enterprise Environment

By the time you're reading this, FreeNAS has been downloaded more than 5.5 million times. For home users, it's become an indispensable part of their daily lives, akin to the DVR. Meanwhile, all over the world, thousands of businesses universities, and government departments use FreeNAS to build effective storage solutions in myriad applications.

What you will learn...

- How TrueNAS builds off the strong points of the FreeBSD and FreeNAS operating systems
- How TrueNAS meets modern storage challenges for enterr

The FreeNAS operating systems is fre
the public and offers thorough doc
active community, and a feature-ric
the storage environment. Based on Free
can share over a host of protocols (SMB
FTP, iSCSI, etc) and features an intuitiv
the ZFS file system, a plug-in system
much more.

Despite the massive popularity
aren't aware of its big brother dut
data in some of the most demand
environments: the proven, enterp
professionally-supported line of

But what makes TrueNAS diffe
Well, I'm glad you asked...

Commercial Grade Suppo

When a mission critical stor
organization's whole operat
halt. Whole community-bas
free), it can't always get an
and running in a timely m
responsiveness and expe
dedicated support team
provide that safety.

Created by the sam
developed FreeNAS.

CYBER SECURITY EXPO

8-9 October 2014
ExCeL London

www.cybersec-expo.com

A NEW event, for a new era of cyber threats

» The most comprehensive analysis anywhere of how to protect the modern organisation from cyber threats

» Free to attend seminars delivered by Mikko Hypponen, Eugene Kaspersky and many more

» Attend the "Hack Den" a live open source security lab to share ideas with White Hat hackers, security gurus, Cyber Security EXPO speakers and fellow professionals

» Network with industry experts and meet with Cyber Security exhibitors

» Discover what the IT Security team of the future will look like

Register NOW

www.cybersec-expo.com

Cyber Security EXPO is the new place for everybody wanting to protect their organisation from the increasing commercial threat of cyber attacks. Cyber Security EXPO has been designed to provide CISOs and IT security staff the tools, new thinking and policies to meet the 21st century business cyber security challenge.

Cyber Security EXPO delves into business issues beyond traditional enterprise security products, providing exclusive content on behaviour trends and business continuity. At Cyber Security EXPO, discover how to build trust across the enterprise to securely manage disruptive technologies such as: Cloud, Mobile, Social, Networks, GRC, Analytics, Identity & Access, Data, Encryption and more.

Co-located at

IP EXPO EUROPE
8-9 October 2014 ExCeL London

www.ipexpo.co.uk

FREE REGISTRATION

Sponsors

ARUBA networks · CITRIX · cloudera · FUSION-IO · Gigamon · hp · NetApp · paloalto · riverbed · Schneider Electric · VEEAM IT JUST WORKS! · vmware

Beyond BIOS, The Extended Firmware Interface (EFI)

This article describes the overall features and principles of the Extended Firmware Interface (EFI), including why you might want to use it, how EFI boots and what types of boot loaders you might use with it to enable non-Windows 8 OSes to boot on an EFI computer. The next three parts of this series will describe how to prepare for the installation of an EFI environment on an Intel-based computer, how to perform the installation and how to manage the computer once it's up and running.

EFI is very different from a PC BIOS, as it offers a wide range of functionality even before the OS starts loading. It is modular (you can add custom code or drivers), runs on various platforms and applications, its drivers can be written in C instead of assembler making them more portable, etc. Besides the native CPU code, EFI supports custom byte code, so drivers can be compiled so that they are portable between CPU architectures without the need for recompilation.

Introduction

Once upon a time, the first IBM PC 5150 was shipped in 1981 with a new 16-bit processor, made by Intel Corporation, and bundled with a firmware known as the *Basic Input Output System* (BIOS). The BIOS was the interface between all hardware devices and the *Operating System* (OS). At the beginning, there was no problem with this approach, but when hard disk and RAM memory prices slowed down, many features supposed a handicap:

- No more than four primary partitions are allowed
- Booting process requires 16-bit real mode
- Boot process starts by loading 512-bytes of data (Master Boot Record, MBR)
- Disks over 2 TB are not supported by BIOS approach
- BIOS is unable to access any disk file system and therefore cannot load any executable image file such as OS kernels

The i386 compatibility architecture was based on keeping the initial bootstrapping process used since 1981. It did not take advantage of protected mode and 32-bit register addressing provided by 80386 and later Intel microprocessors. It was not modified until 2005 at which time the *Extensible Firmware Interface* (EFI) was developed to provide a more versatile and updated boot process based on the ability to load and execute ELF images directly from the initialization code.

In reference to hard disk devices, BIOS-based computers could only handle up to 232 sectors using 512-byte sectors. This leads to a 2 TB limit on storage capacity. Besides, the special partition managed by EFI and termed *EFI Special Partition* (ESP) can use both the FAT-32 file system, as encouraged by EFI Standard, and FAT-16.

It can even use HFS+ for *Mac OSx* computers. This ESP has the partition code 0xEF00, which allows a quick identification.

As the BIOS cannot access a file system on a disk and therefore is unable to load an executable image file such as OS kernels, every OS must have its own boot loader using the BIOS approach, which constitutes a huge source of problems. A way to avoid the use of separate boot loaders for each OS installed, is to use *Multi-boot Specification* (MS) which will be covered in another article.

Considering the historical background explained above, the *Extensive Firmware Interface* (EFI) has its roots in 1998 with the *Intel Boot Initiative* (IBI) program. Hence, the EFI specification, which has been developed and supported by a consortium integrated by Intel and Microsoft, among other companies, defines an API and data structures to handle generic firmware in a wide variety of platforms in order to provide OS loaders, EFI device drivers, and diagnostics by means of an EFI command interpreter or EFI Shell.

The first EFI specification was EFI 1.02 released in 2000. Due to legal issues, it was re-released two years later under the denomination EFI 1.10 and restricted to Itanium microprocessors. In order to avoid undesirable scattering, the Unified EFI Forum was created including companies such as Intel, AMD, AMI, Apple, Dell, HP, IBM, Phoenix and, of course, Microsoft, among others. This initiative led to Universal EFI (UEFI) standardization. Later on, AMD created its own 64-bit architecture, AMD64, which was backward compatible with IA32. The AMD64 architecture is equivalent to Intel's EM64T architecture and it was eventually supported in the UEFI 2.0 standard. Nowadays, the latter standard is UEFI 2.1 which includes a few changes regarding its predecessor.

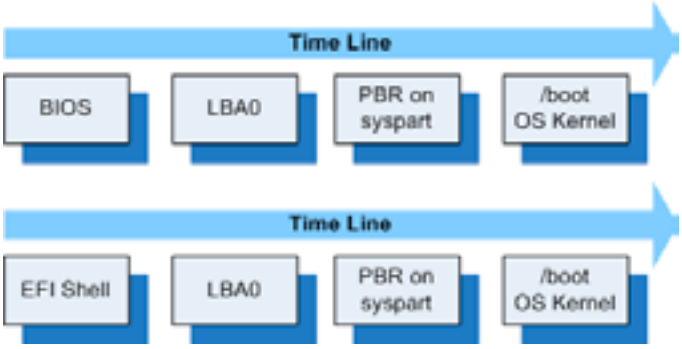


Figure 1. *Comparison between BIOS and UEFI approach*

Last but not least, EFI can boot a computer faster than BIOS-based booting. On average, the EFI booting process is more than 20 seconds faster than using BIOS boot mode.

EFI has its drawbacks too, of course. The most important of these is the fact that it's new. This means that old boot loaders don't work with it and users are unfamiliar with it. Another significant problem is that the EFI boot process assumes the OS will run in the same bit depth as the EFI. Because all UEFI-based PCs and most EFI-based Macs use 64-bit firmware, this means that 64-bit OSes work best with these computers (*the earliest Intel-based Macs used 32-bit EFIs though*). Installing a 32-bit version of Linux on a computer with a 64-bit EFI is possible, but you'll give up runtime EFI interfaces. This makes boot loader maintenance harder, since the efibootmgr utility (which will be described in part three of this series) relies on such interfaces. For this reason, I recommend installing a 64-bit distribution if your firmware is 64-bit.

GUID Partition Table (GPT)

The *GUID Partition Table* (GPT) is a new standard for disk partitioning providing a set of advanced features such as:

- Modern logical block addressing (LBA)
- 64-bit LBA pointers to manage partitions up to 8 ZB
- Support for non-512 byte sector size disks
- Up to 128 partitions per disk
- Inclusion of backup partition table

Although *NetBSD* can access GPT disks by using *dk-wedges*, it is not possible to boot off a GPT disk in a straightforward manner and the current strategy to boot is similar to EFI bootstrapping:

```
BIOS â†’ LBA0 â†’ PBR on EFI syspart â†’ /boot â†’ NetBSD
    kernel
```

Secure Boot and Microsoft Legacy

One of the most controversial features of EFI is *Secure Boot*. This feature was originally intended to improve security by ensuring that only boot loaders signed with a crypto key can run. In such a way, malware code cannot be executed as it is not signed with this key. However, Microsoft requires Secure Boot enabled for Windows 8 use in desktop and laptop computers and as a practical matter, Microsoft's keys are included in the vast majority of new computers with *UEFI* support. No other company/organization has the power to guarantee that their keys are also included.

The only way to bypass this inconvenience is through the use of Microsoft's signing service. Otherwise, the only way to avoid any issues with non-Windows OSes is to disable Secure Boot, which is perfectly possible if you do not want to use MS Windows 8.

A Tour on EFI Shell

Before introducing the main topic, it is important to take a preliminary approach on *UEFI* usage so that readers can see the main differences between the former BIOS and the new paradigm for new 64-bit Intel computers. There are many possibilities.

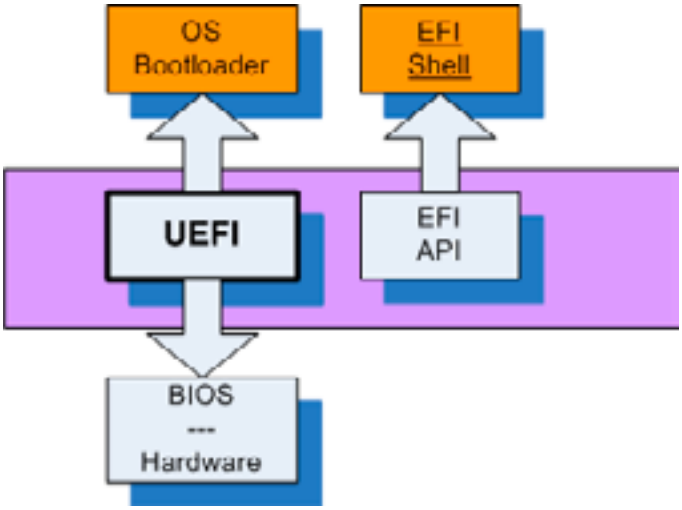


Figure 2. *BIOS and UEFI End-User Interface*

End-User EFI Commands

Whenever a new Intel-based computer is started, the UEFI program starts its execution to get into a shell as follows:

```
fs0:\> ver
EFI Specification Revision : 1.10
EFI Vendor : INTEL
EFI Revision : 14.62

 fs0:\> ls
 fs0:\> devices -b
fs0:\> dh -b
fs0:\> cd apps
fs0:\apps> ls
fs0:\apps> load tcpipv4.efi
fs0:\apps> ifconfig -a
fs0:\apps> ifconfig lo0 inet 127.0.0.1 up
```

For MacOS X computer's owners, the following command is useful to examine GPT hard disk.

```
fs0:\diskutils> diskpart
```

Eventually, if you need some help, issue the following command to display commands one screen at a time:

```
help -b
```

Programming EFI

If you are an experienced programmer, it is possible to develop and evaluate your own EFI applications even using an IA-32 computer. To get the true flavor of EFI, you need two different environments:

Runtime Environment

To explore EFI on IA-64 or in IA32 computers by using a BIOS32 boot floppy provided by Intel to boot into a real EFI environment running x86 with legacy BIOS.

Development Environment

To develop EFI programs, such as device drivers, boot loaders and so on, consisting of:

- A host operating system.
- GNU CC toolchain
- Intel EFI Application Toolkit

Intel EFI Application Toolkit is provided by *TianoCore* project and is available under BSD licenses in *www.tianocore.org*. Alternatively, there is a GNU EFI development port but it is not mature enough yet.

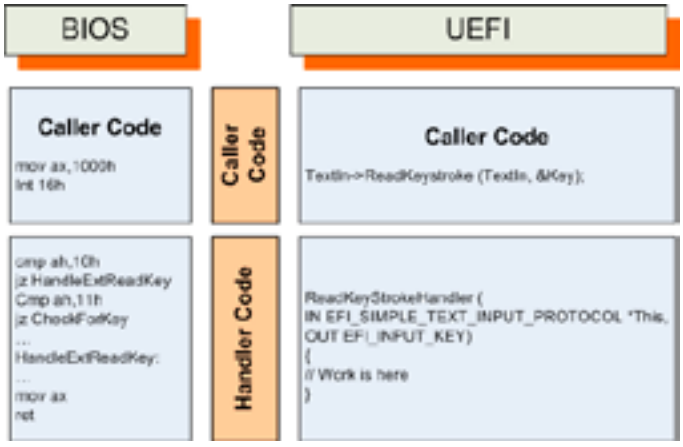


Figure 3. *BIOS vs. UEFI API Program Development*

In order to compare the main differences between BIOS and UEFI, be aware that UEFI offers a complete API to support low-level firmware development.

EFI Boot-Loaders

In comparison with BIOS boot loaders, EFI boot loaders are still under heavy development as follows:

- ELILO: one of the most reliable bootloaders for GNU/Linux systems but it requires the kernel to be loaded from ESP and does not allow other locations.

- GRUB-2: supports both BIOS/EFI booting but requires installing an EFI-capable package such as grub-efi. Regarding its predecessor, GRUB Legacy does not support the EFI booting process. GRUB-2 is sometimes very complex to handle.
- rEFIt: is not capable of booting a kernel directly and requires a chainload to make it possible.

My only experience at the moment with Linux kernels is that work is being done to embed EFI boot loader support to load the kernel directly without using a third-part EFI boot loader. To sum up, have a look at the following table:

Boot loader	Load Linux	Kernel location	Chain load
ELILO	Y	ESP	N
GRUB-2	Y	any	Y
rEFIt	N	N/A	Y
Linux kernel	Y	ESP	N

And just to conclude this section, one of our favorite UEFI boot loaders, and our recommended choice for Mac OS X fans, is *rEFIt*, which supports graphical output as shown:

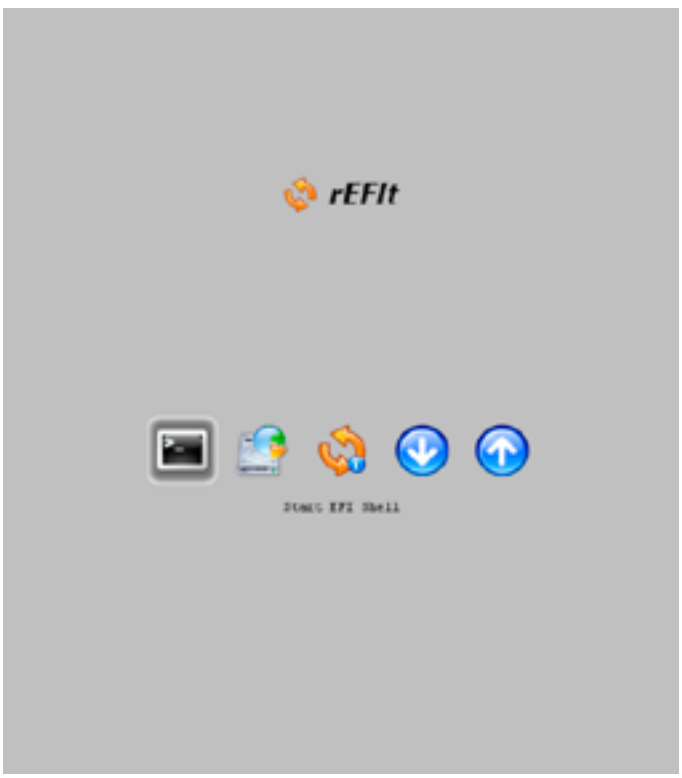


Figure 4. *rEFIt UEFI bootloader main screen*

No matter which UEFI boot loader you choose, have a clear understanding about the implications of using it,

especially if you have to coexist with *Microsoft™ Windows OS* on your computer.

NetBSD/EFI in I386 Architectures

The *NetBSD/i386* boot process uses a two-stage boot loader where the first stage is installed in a well-known physical location (the first sector of the disk *MBR*) and this stage provides the necessary information to start with the second stage boot loader placed on the root file system and transfer the control to it. Once the second stage boot loader has taken control, it swaps the processor into the protected mode with a full 32/64-bit addressing and no segmented memory as in 16-bit real mode.

NetBSD and GPT Awareness

It's possible to boot *NetBSD* from the *GPT partitioned* disk when using a PC BIOS computer. The approach is quite similar to the NetBSD MBR boot loader and is comprised of three parts:

- `mbr_gpt/mbr_gpt_com0` is an LBA0 loader intended to be used by a BIOS-based computer, whose main aim is to find a bootable *GUID* partition.
- `bootxx_fat16` is a PBR loader which can be executed by either MBR loader or BIOS. The mission of this function is to load the NetBSD boot(8) kernel program and put the loader into an ESP FAT16 partition, which can be a source of trouble, due to the recommendation of using ESP FAT32-formatted partitions.
- `NetBSD boot(8) kernel loader` is in charge of loading and running the NetBSD kernel from either a GUID or disklabel partition.

At the moment of writing this article, an effort is being done to get an EFI boot loader for *NetBSD* systems in order to get rid of the former GRUB-based approach. It will be available by the end of this year, I hope. Anyway, if you are not using MS Windows 8 in your computer, you can safely following the instructions to minimize the impact of new EFI-based computers.

Installation Procedure of NetBSD

To build and install the NetBSD loader, you should have the following software:

- Current NetBSD kernel sources as distributed in sys-src package.
- sbin/gpt and usr.sbin/installboot loader installation tools.
- Ensure you have the latest GPT bootloader patch available at *http://www.netbsd.org/~mishka/gptboot/gptboot.patch*

The following steps describe what you'd need to build a NetBSD loader:

1. Download all the sources above
2. Prepare `src tree` for patching by making directories missing. See the list of the directories inside the patchfile:

```
$ awk '/^WARNING:/ {print $(NF)}' gptboot.patch
```

3. Apply the patch
4. Build GPT loaders and tools at the following directories:

```
sys/arch/i386/stand/mbr/mbr_gpt
sys/arch/i386/stand/fatboot/fat16
sys/arch/i386/stand/boot/biosboot
sbin/gpt
usr.sbin/installboot
```

All of above builds just fine on NetBSD 5.0 (including amd64) without cross compilation. The install boot may require passing -DSMALLPROG to make [1] to exclude extra stuff and simplify the build process. Also, new `sys/sys/bootblock.h` has to be used in place of `/usr/include/sys/bootblock.h`.

Loader Installation

First of all, you should prepare your disk. The disk should be GPT partitioned and have at least two partitions, one for NetBSD, and one for boot loader as follows:

```
# gpt create sd0
# gpt add -s 65536 -t efi sd0
# gpt add -t ffs sd0
# gpt show
```

then issue a set of `dkctl addwedge commands` or reattach the disk to configure dk wedges automatically:

NB: Wedges are not supported on vnd(4) devices
Next: Format partitions accordingly:

```
# newfs_msdos -F 16 /dev/dk0
# newfs /dev/dk1
```

Please note that *newfs_msdos* seems to have a bug and can incorrectly determine file system size (*check number of file sectors reported by newfs_msdos; it must be less or equal to partition size*). If it appears, please reformat file system explicitly specifying correct fs size via newfs_msdos -s option.

To install the loaders, issue the following commands:

```
# .../gpt biosboot -c $NETBSDSRC_DIR/sys/arch/i386/stand/
  mbr/mbr_gpt/mbr_gpt sd0
# .../installboot /dev/rdk0 $NETBSDSRC_DIR/sys/arch/i386/
  stand/fatboot/fat16/bootxx_fat16
# mount -t msdos /dev/dk0 /mnt
# cp $NETBSDSRC_DIR/sys/arch/i386/stand/boot/biosboot/
  boot /mnt
# echo "menu=Boot NetBSD:boot hd0b:netbsd"
  > /mnt/boot.cfg
# umount /mnt
```

The gpt(8) will automatically find the EFI *system partition* on sd1 and instruct mbr_gpt where to load PBR from. But rest loaders should be installed on dk wedges. If you're confused about the names, you may use gpt(8) on a wedge, but in this case mbr_gpt will load PBR from the specified wedge:

```
# .../gpt biosboot -c $NETBSDSRC_DIR/sys/arch/i386/stand/
  mbr/mbr_gpt/mbr_gpt dk0
```

Then, install kernel and base system files through:

```
# mount /dev/dk1 /mnt
# cp /netbsd /mnt
```

Then, create the usual NetBSD hierarchy: `/dev`, `/etc`, `/sbin`, etc... and specify the root partition as it will be enumerated by DKWEDGE_AUTODISCOVER:

```
# echo "/dev/dk1 / ffs rw 1 1" > /etc/fstab
# umount /mnt
```

Please note the disk names on previous steps. It might be somewhat confusing, so here is an explanation:

- gpt biosboot ... sd0: LBA0 means installation in the very first sector of the physical disk, so we should specify the parent device of our GPT wedges. A dk device can also be used. In that case, mbr_gpt will look for a GUID partition matching the dk device at the moment of installation.
- install boot `/dev/dk0 ...: bootxx_fat16` should be installed onto the EFI System Partition. See gpt add commands earlier. mount -t msdos `/dev/dk0` ...: the boot(8) should be stored on the EFI System Partition as well.
- echo `menu=Boot NetBSD: boot hd0b:netbsd` ...: the hd0b means the second partition, which matches dk1 after boot. Please see the to-do list about that. The resting commands refer to dk1 which is the NetBSD FFS partition.

Now! Reboot and Have fun. :)

The Easy Way, Using another OS boot-Loader to Start the BSDthe OS

The fastest and safest way to start a BSD OS like NetBSD is to *use another operating system with full EFI support such as GNU/Linux* in order to use its own boot loader GRUB as our boot loader for a wide variety of non-Microsoft Windows OSes.

1. Install an EFI-compliant GNU/Linux Distribution for x86-64 bits. I strongly recommend GNU Debian 7.5 IA64.
2. Ensure that GNU Debian has its own ESP. By default, this ESP is 200 MB size.
3. Power on the UEFI-compliant computer by pressing down the key "Supr" to stop the default booting process.
4. Enter and execute the grubx64.efi EFI application using the built-in UEFI shell.
5. Select the desired OS bootloader in the GRUB-2 menu.

That is all. If you do not want or are not happy with dealing with complexity, this is the best alternative to take advantage of the new *UEFI* PC architectures and to get rid of the BIOS the old-fashioned way.

Dealing with the SecureBoot Feature (SHIM boot Loader)

In this sense, despite Microsoft's efforts to make our lives more difficult by means of the SecureBoot feature, thanks to the work of ... a functional version of an EFI boot loader named SHIM which is available for download at *http://www.codon.org.uk/~mjg59/shim-signed/*. The procedure to use it cannot be simpler than the following steps:

- Rename "shim.efi" to "bootx64.efi".
- Put this file into '/boot/EFI directory'.

Now, generate a certificate and put the public half as a binary DER file somewhere on your install media. On boot, the end-user will be prompted with a 10-second countdown and a menu. Choose "Enroll key from disk" and then browse the file system to select the key and follow the enrolment prompts. Any boot loader signed with that key will then be trusted by shim, so you probably want to make sure that your grubx64.efi image is signed with it.

This design has been borrowed from Suse's boot loader developers and requires that the boot loader itself has its own key database, distinct from the one provided by UEFI specification. In such a way, as the boot loader is in charge of its own key enrolment, the boot loader has the freedom to manage its own policy.

Testing UEFI

In order to avoid any damage to a real computer, we strongly recommend you use a virtualized environment to test any *UEFI* features before moving on to the real computer as follows:

```
qemu-system-x86_64 -serial stdio -bios OVMF.fd -hda
   fat:<path to boot directory>
qemu-system-x86_64 -serial stdio -bios OVMF.fd -cdrom
   <path to ISO image>
```

Also, *FreeBSD* developers have documented the way of creating UEFI media for testing purposes. To wrap up, let us describe the way of creating a USB HD and CD-ROM UEFI capable media:

CD-ROM with UEFI support media generation

```
gpart create -s gpt da0
gpart add -t efi -s 800K da0
gpart add -t freebsd-ufs da0
dd if=/boot/boot1.efifat of=/dev/da0p1
newfs /dev/da0p2
```

Then, perform the install to the UFS partition, as usual:

```
mount /dev/da0p2 /mnt
make DESTDIR=/mnt installkernel installworld distribution
echo "/dev/da0p2 / ufs rw 1 1" >> /mnt/etc/fstab
umount /mnt
```

USB HD with UEFI support media generation

```
> dd if=/dev/zero of=efiboot.img bs=4k count=100
> mdconfig -a -t vnode -f efiboot.img
> newfs_msdos -F 12 -m 0xf8 /dev/md0
> mount -t msdosfs /dev/md0 /mnt
> mkdir -p /mnt/efi/boot
> cp loader.efi /mnt/efi/boot/bootx64.efi
> umount /mnt
> mdconfig -d -u 0
> makefs -t cd9660 -o bootimage='i386;efiboot.img' -o
   no-emul-boot -o rockridge -o label="UEFItest" -o
   publisher="test" uefi-test.iso image
```

Remember

The boot directory must contain the EFI executables required.

Conclusions and Remarks

If you are wondering why the BIOS approach was kept for decades, a justification for such longevity may be found in the fact that MS-DOS for PC was built on top of the BIOS and MS-DOS programs called BIOS routines through software interrupts. Hence, the BIOS disk I/O routine corresponds to INT 13h. In order to preserve compatibility, this approach survived an unexpectedly long time, despite its technical weaknesses and limitations.

What's more, EFI was originally designed for Itanium 64-bit processors although nowadays, IA-32 may support EFI-based firmware and there are some companies shelling IA-32 computers with full EFI support, such as Inside Software.

Furthermore, the BIOS depends on VGA which is a legacy standard and does not allow defining new boot devices unless they've already been included in BIOS routines. The current approach for graphics support is UGA, which is provided by EFI too. In such a way, the UEFI approach constitutes a true extensible firmware management system.

References

[1] Official UEFI documentation, *www.uefi.org*
[2] Official ELILO, *elilo.sourceforge.net*
[3] rEFIt boot loader, *refit.sourceforge.net*

Acronyms and Abbreviations

MBR	Master Boot Record
MS	Multi Boot Specification
ESP	EFI System Partition
EFI	Extended Firmware Interface
IBI	Intel Boot Initiative
UEFI	Universal Extended Firmware Interface
GPT	GUID Partition Table
VGA	Video Graphics Adapter
UGA	Universal Graphics Interface

JOSÉ B. ALÓS

José B. Alós has developed an important part of his professional career since 1999 as an EDS employee, as UNIX System Administrator, mainly focused on SunOS/Solaris, BSD and GNU/Linux. Five years ago he joined EADS Defense and Security, nowadays CASSIDIAN as a responsible for providing support for end-users in aircraft engineering departments for long-term projects. That is the main reason underneath this article as VAX/VMS systems still play a paramount role in today's aerospace industry for a wide variety of embedded RT systems conceived for mission and flight operations. He was also Assistant Professor in the Universdad de Zaragoza (Spain), specialized in the design of High Availability solutions and his academical background includes a PhD in Nuclear Engineering and three MsC in Electrical and Mechanical Engineering, Theoretical Physics and Applied Mathematics.

Great Specials

On FreeBSD® & PC-BSD® Merchandise

Give us a call & ask about our
SOFTWARE BUNDLES
1.925.240.6652

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

$29.95
PC-BSD 9.1 DVD

$49.95 Save a BUNDLE!
The PC-BSD 9.0 Users Handbook
PC-BSD 9.1 DVD

$99.95
The FreeBSD CD or DVD Bundle
Inside each CD/DVD Bundle, you'll find:
FreeBSD Handbook, 3rd Edition
Users Guide FreeBSD Handbook, 3rd Edition, Admin Guide
FreeBSD 9.1 CD or DVD set
FreeBSD Toolkit DVD

Stylish Dress Attire
Look Your Professional Best

Comfy Apparel
Stay Warm in Zip Ups & Pullovers

T-Shirts
Lots of Styles to Choose From

FreeBSD 9.1 Jewel Case CD/DVD $39.95

CD Set Contains:

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

FreeBSD 9.0 CD $39.95
FreeBSD 9.0 DVD $39.95

FreeBSD Subscriptions

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

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

PC-BSD 9.1 DVD (Isotope Edition)

PC-BSD 9.1 DVD $29.95
PC-BSD Subscription $19.95

The FreeBSD Handbook

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

The FreeBSD Handbook Specials

The FreeBSD Handbook, Volume 2 (Both Volumes) $59.95
The FreeBSD Handbook, Both Volumes & FreeBSD 9.1 ..... $79.95

PC-BSD 9.0 Users Handbook $24.95

BSD Magazine $11.99

The FreeBSD Toolkit DVD $39.95

FreeBSD Mousepad $10.00

FreeBSD & PCBSD Caps $20.00

BSD Daemon Horns $2.00

Bundle Specials!
Save $$$

Just Plain Fun
Mousepads & Novelty Horns

BSD Magazine
Available Monthly

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

Debugging and Troubleshooting: Fun, Profit and Go Home Earlier

Debugging/Troubleshooting is a really useful skill when you are working on maintaining legacy applications doing some small incremental changes to an old code base, where the code has been touched by so many hands over the years and it is really becoming a mess. So, management has decided that the code works as-is and you are not allowed to change it all over "the right way (tm)".

In this tutorial, I'm going to represent a real life situation where debugging skills will save us time, headaches and to possibly find a solution using a minimal amount of effort.

First, we are going to debug an old legacy C application that takes plain text files and inserts them into a database until some new change made some dormant "feature" available for the users (data is getting truncated and users are complaining, this needs to be fixed ASAP before close of business). For this situation we are going to use the classic "gdb" debugger.

Second, we are going to debug a Java application that is having performance issues (takes 4 times as the old C application) and it is the new way of doing things instead of the old C application that gets the data into the database; also unfortunately, believe it or not, it has the same issue as their old C counterpart.

We will use heap dumps, *jdb* (*http://docs.oracle.com/javase/7/docs/technotes/tools/windows/jdb.html*) and *gdb* for this.

In the third scenario, we will go back to the first one but we will take a different approach. We will debug without having the source code and only relay in the disassembled code we could see through *gdb*.

For the Fourth scenario, we will only have a heap dump and we will need to go all the way to find the issue lurking in the java code.

Finally, we will approach both situations using *Dtrace* which is available in *FreeBSD*, *OSX*, *Solaris* and *OpenSolaris* and we will check if this tool is beneficial and a time saver in the process.

First Scenario

For debugging, we will use *GDB*, if you don't have the ports collection installed then you should do so (we need *postgresql* for this tutorial so you should use ports if you want an up to date version of postgresql), using the following instructions as root:

```
# portsnap fetch
```

When running Portsnap for the first time, extract the snapshot into `/usr/ports` as follows:

```
# portsnap extract
```

After the first use of *Portsnap* has been completed as shown above, `/usr/ports` can be updated as needed by running:

```
#portsnap fetch
# portsnap update
```

When using fetch, the extract or the update operation may be run consecutively, like so:

```
# portsnap fetch update
```

We will need the following packages for this tutorial:

- postgresql-client
- postgresql-server

You could install version 8.4.21 using pkg (*http://www.freebsd.org/cgi/man.cgi?query=pkg&sektion=7*) as root using the following commands:

```
pkg install postgresql84-client-8.4.21
pkg install postgresql84-server-8.4.21_1
```

For detailed instructions on installing and configuring *postgresql*, you should read one of the guides from the official site: *https://wiki.postgresql.org/wiki/Detailed_installation_guides#FreeBSD*.

Also we will use some test data from *http://www.briandunning.com/sample-data/* to execute the examples in this tutorial. We will need to download the US 500 sample (this one is free). you should rename this file to us.csv for the purposes of this tutorial.

Once you have postgresql installed, you need to create the *us table* using the us.sql (All the files needed for this tutorial are in the *workshop.tar.gz* included).

The Incident (a.k.a Production Down)

You have received an email stating that the current process for adding new clients has stopped working and nobody knows why, all that management knows is that the file must be loaded to the database before business closes or they will have to give upper management a serious explanation about what happened and how they will prevent this from happening in the future. So, to avoid all this stress you have been selected to fix this problem right away, and before anyone knows what is actually happening (which means you need to work all night as needed). So let's get to it as quickly as possible as we don't want to spend our nap time debugging an old application that we really don't want to touch as there is no documentation nor the original programmers are available. As far as you know, this program was a product of a joint venture between contractors of different nationalities. Also the code and comments are in Spanish.

Let's start running the program. It takes as a parameter a file name where the client data is: see Figure 2.

All seems fine, the program does what it is supposed to do, false alarm again just 30 minutes to go home. Just to be sure, I'll check the table to see if all the data is in

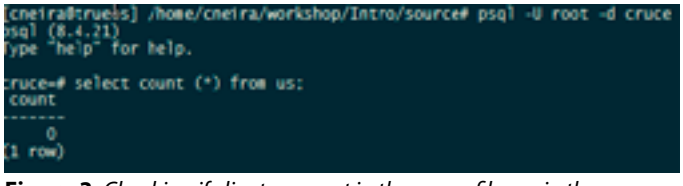


Figure 3. *Checking if clients present in the us.csv file are in the database*

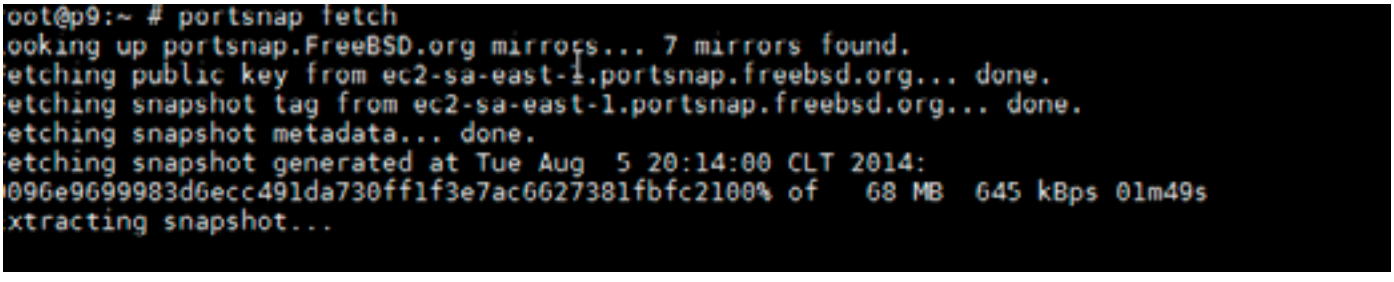


Figure 1. *Portsnap fetch*

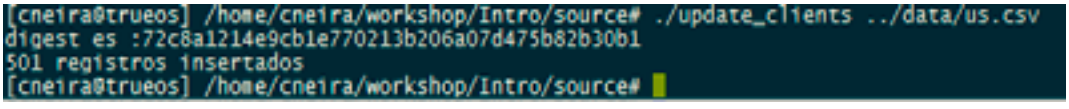


Figure 2. *Trying to update clients using the us.csv file*

there; after all, that is the issue that has been reported (see Figure 3).

Tough luck, there is really an issue in here. I'll fetch the source code and fire up *gdb*. I have better things to do than debug old code all night and according to management, this one must be fixed before the next run as the users are inserting the data manually.

The source code fortunately was still on the backups, so I created a minimal "makefile" for this. I just needed this one to compile and let the compiler put all the debugging symbols needed in the object file for an "easy" debug session.

```
#This one updates the DB with the clients taken from a plain text file
All     :       update_clients
update_clients: update_clients.o
        cc -ggdb update_clients.o -L /usr/local/lib -lpq -o update_clients

update_clients.o : update_clients.c
        cc -ggdb -c update_clients.c  -I /usr/local/include

clean   :
          rm  *.o
          rm  update_clients
```

Figure 4. *Simple makefile to start debugging*

The "-ggdb" flag is an old *gcc* flag that does the following according to the official manual (*https://gcc.gnu.org/onlinedocs/gcc-4.7.4/gccint/All-Debuggers.html*):

When the user specifies -ggdb, GCC normally also uses the value of this macro to select the debugging output format, but with two exceptions. If DWARF2_DEBUGGING_INFO is defined, GCC uses the value DWARF2_DEBUG. Otherwise, if DBX_DEBUGGING_INFO is defined, GCC uses DBX_DEBUG.

If we are using clang, this does not matter as you can use the -g flag "Generate complete debug info". Let's compile this thing and see what is happening: see Figure 5.

Well, at least it compiles, it could be worse at this time. Now, the debug symbols are in there so let's try setting some breakpoints to catch the problem at hand, looking at the source code the first obvious breakpoint must be set in the insert function call: see Figure 6.

I'll start a debugging session. I'll pass as parameter the us.csv file to the program as follows: see Figure 7. To start a debugging session, just type the following command:

```
gdb <program name>
```

This will take us to a (*gdb*) prompt where we could use all the commands available in the GDB debugger. If we were to debug a running program, we should type:

```
(gdb) attach <pid of running program>
```

And it will take us to the same prompt again, but notice that in this case it will cause the world to stop for the running program until we let it complete in our debug session. Also if we had a core dump, we could check the stack trace, but in this case we have no *coredump*

```
cc -ggdb -c update_clients.c  -I /usr/local/include
update_clients.c:130:6: warning: implicit declaration of function 'valida_cruce_ab' is invalid in C99 [-Wimplicit-function-declaration]
 if(!valida_cruce_ab()) { return 0; };
     ^
update_clients.c:205:5: warning: implicit declaration of function 'insert_hash' is invalid in C99 [-Wimplicit-function-declaration]
    insert_hash(sha_hash(filepath));
    ^
update_clients.c:308:84: warning: format specifies type 'long' but the argument has type 'char *' [-Wformat]
    if(atol(ab.saldo_disponible) < 0) { printf("saldo disponible negativo (%ld)\n",ab.saldo_disponible); ERR= 0; }
                                                                                  ~~~                  ^~~~~~~~~~~~~~~~~~~~~
                                                                                  %s
update_clients.c:309:80: warning: format specifies type 'long' but the argument has type 'char *' [-Wformat]
    if(atol(ab.saldo_contable) < 0) { printf("saldo contable negativo [%ld]\n",ab.saldo_contable); ERR= 0; }
                                                                               ~~~                  ^~~~~~~~~~~~~~~~~~~~
                                                                               %s
update_clients.c:397:22: warning: field width should have type 'int', but argument has type 'off_t' (aka 'long') [-Wformat]
        sprintf(databuff,"%*.*s",sbuf.st_size,sbuf.st_size,data);
                  ^~~  ~~~~~~~~
update_clients.c:412:30: warning: passing 'int *' to parameter of type 'unsigned int *' converts between pointers to integer types with
      different sign [-Wpointer-sign]
    EVP_DigestFinal(&ctx,ret,&md_len);
                             ^~~~~~~
/usr/include/openssl/evp.h:565:69:       passing argument to parameter 's' here
int     EVP_DigestFinal(EVP_MD_CTX *ctx,unsigned char *md,unsigned int *s);
                                                                    ^
update_clients.c:423:12: warning: returning 'unsigned char [64]' from a function with result type 'char *' converts between pointers to
      integer types with different sign [-Wpointer-sign]
    return ret;
           ^~~
update_clients.c:430:1: warning: type specifier missing, defaults to 'int' [-Wimplicit-int]
insert_hash(char* hash)
^~~~~~~~~~~
update_clients.c:436:1: warning: control reaches end of non-void function [-Wreturn-type]
}
^
9 warnings generated.
cc -ggdb update_clients.o -L /usr/local/lib -lpq -o update_clients
[cneira@trueos] /home/cneira/workshop/Intro/source#
```

Figure 5. *Makefile output*

```
194 void read_file_and_insert(char* filepath,char* szTable)
195 {
196
197     FILE* fp;
198     char  line[256];
199     int count=0;
200     char tipocruce[3];
201
202     bzero (line,sizeof line);
203
204     if (!(fp=fopen(filepath,"r"))) {perror("no se encontro archivo\n"); exit(1);}
205     insert_hash(sha_hash(filepath));
206     if(!memcmp(filepath,"AB",2))sprintf(tipocruce,"%.2s","AB");else sprintf(tipocruce,"%.2s","CD");
207
208 //young intern guy 09-01-2005:  I'll create a temporary  us table, as the all have the same format  I'll change it later to a table
country
209  if(!memcmp(filepath,"us",2)) sprintf(tipocruce,"%.2s","US");
210 // contractor 1 09-01-2007 : I'll add the ca client to the existing us table that is currently being used
211  if(!memcmp(filepath,"ca",2)) sprintf(tipocruce,"%.2s","US");
212 // young intern 2 ; 10-10-2008 : adding new clients  yihaaaa
213  if(!memcmp(filepath,"au",2)) sprintf(tipocruce,"%.2s","US");
214  if(!memcmp(filepath,"uk",2)) sprintf(tipocruce,"%.2s","US");
215 // Wally 10-10-2009 code reuse technique http://search.dilbert.com/comic/Code%20Reuse
216  if(!memcmp(filepath,"au",2)) sprintf(tipocruce,"%.2s","US");
217  if(!memcmp(filepath,"cl",2)) sprintf(tipocruce,"%.2s","US");
218  if(memcmp(filepath,"AB",2)) sprintf(tipocruce,"%.2s","US");
219  if(memcmp(filepath,"CD",2)) sprintf(tipocruce,"%.2s","US");
220 //Wally 10/10/2009
221 //09-01-2005
222      while(!feof(fp))
223          {
224              if( fgets(line,sizeof line,fp)!=NULL)
225                  if(insert (szTable, line,tipocruce))
226              count++;
227          }
228      printf("%d registros insertados\n",count);
229 }
230
231
232
```

Figure 6. *Setting first breakpoint*

```
[cneira@trueos] /home/cneira/workshop/Intro/source# ls
Makefile        update_clients   update_clients.c update_clients.o
[cneira@trueos] /home/cneira/workshop/Intro/source# gdb update_clients
GNU gdb 6.1.1 [FreeBSD]
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "amd64-marcel-freebsd"...
(gdb) b insert
Breakpoint 1 at 0x401890: file update_clients.c, line 125.
(gdb) r us.csv
Starting program: /usr/home/cneira/workshop/Intro/source/update_clients us.csv
[New LWP 100538]
[New Thread 802006400 (LWP 100538/update_clients)]
no se encontro archivo
: No such file or directory

Program exited with code 01.
(gdb) r data/us.csv
Starting program: /usr/home/cneira/workshop/Intro/source/update_clients data/us.csv
[New LWP 100382]
[New Thread 802006400 (LWP 100382/update_clients)]
no se encontro archivo
: No such file or directory

Program exited with code 01.
(gdb) r ../data/us.csv
Starting program: /usr/home/cneira/workshop/Intro/source/update_clients ../data/us.csv
[New LWP 100366]
[New Thread 802006400 (LWP 100366/update_clients)]
Digest es :72c8a1214e9cb1e770213b206a07d475b82b30b1
[Switching to Thread 802006400 (LWP 100366/update_clients)]

Breakpoint 1, insert (szTable=0x402fe7 "AB",
    Data=0x7fffffffd7f0 "first_name,last_name,company_name,address,city,county,state,zip,phone1,phone2,email,web\r\n",
    tipocruce=0x7fffffffd7e9 "US") at update_clients.c:125
125         q= (char*) malloc((strlen(Data)+1) *5);
Current language:  auto; currently minimal
(gdb)
```

Figure 7. *Running gdb using us.csv as parameter to the update _ clients program*

to take a look. Now first, I'll set a breakpoint using the break command and I could type just "b" as a short form, if we need some help with a command, we just type the usual:

```
(gdb)  help <command>
```

For example type "help break" and this screen will be presented to us:

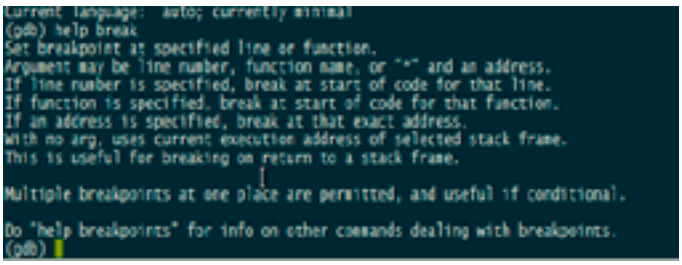


Figure 8. *Running gdb help command*

Now, I have my breakpoint ready at the insert function. I'll run the program and check what will be happening at runtime. I run the program by typing:

```
(gdb) r  ../data/us.csv
```

Where "r" is the abbreviated form of run. You could pass the parameters to the program next to the command. In this case this program only takes one that is the file containing client data (`../data/us.csv`): see Figure 9.

As I'm lucky to have the source code, I can use the *win command*, that will display the source code and the exact line the execution has stopped at as follows: see Figure 10.

Then, we need to check the value of the input parameter data to the insert function, so I just type:

```
(gdb) display  Data
```

This command will print at every time, we hit a breakpoint the value of the Data variable, as long as the breakpoint is within the scope of this variable as follows: see Figure 11.

All seems OK with the data and the functions that make up the sql statement. So, we need to check where the SQL statement is executed. Hence, we will put a break at the query(char*) function, looking at the documentation for

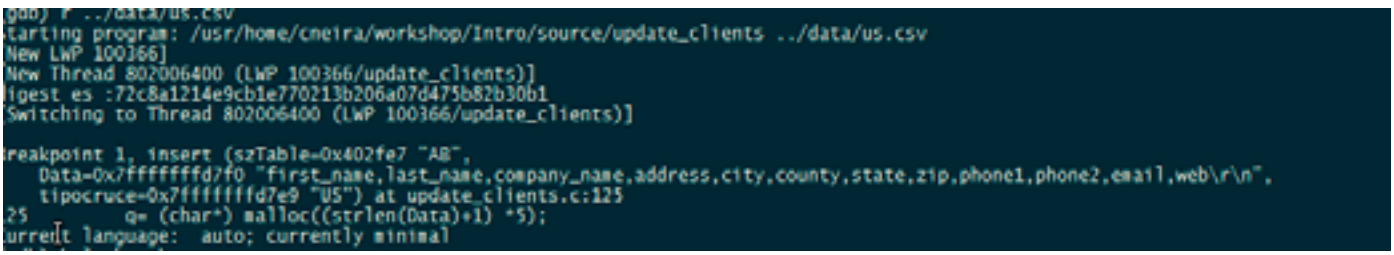


Figure 9. *Running gdb stopping at a breakpoint*

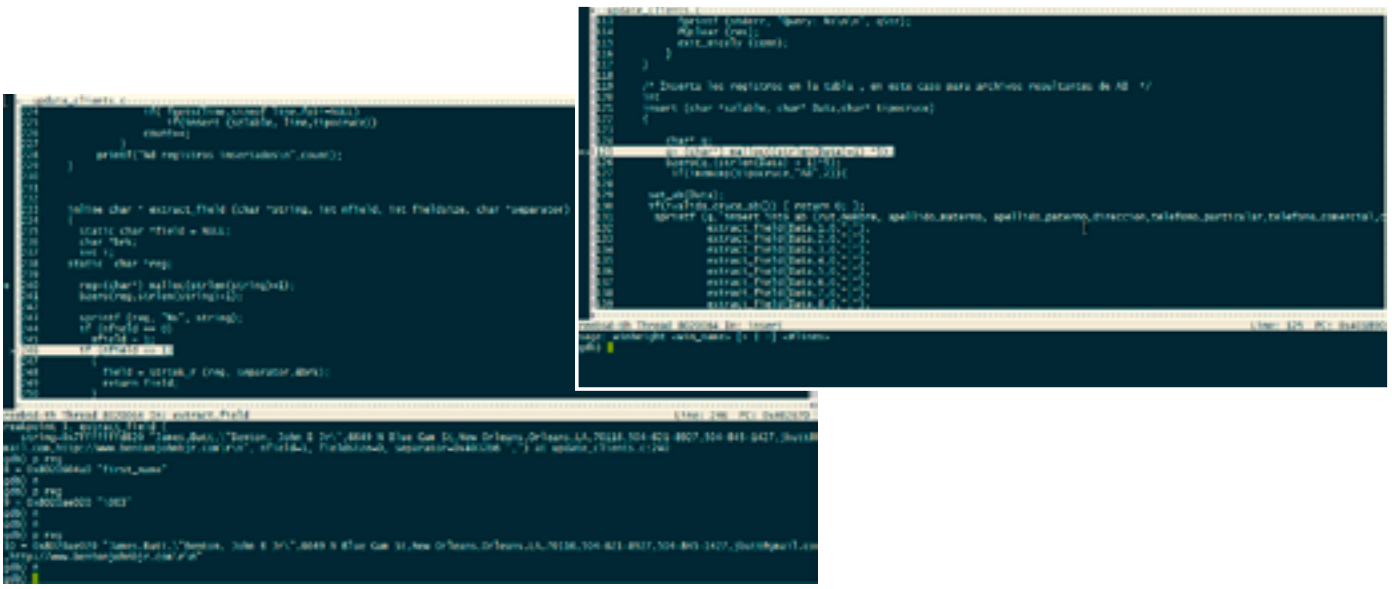


Figure 10. *Running gdb using the win command*

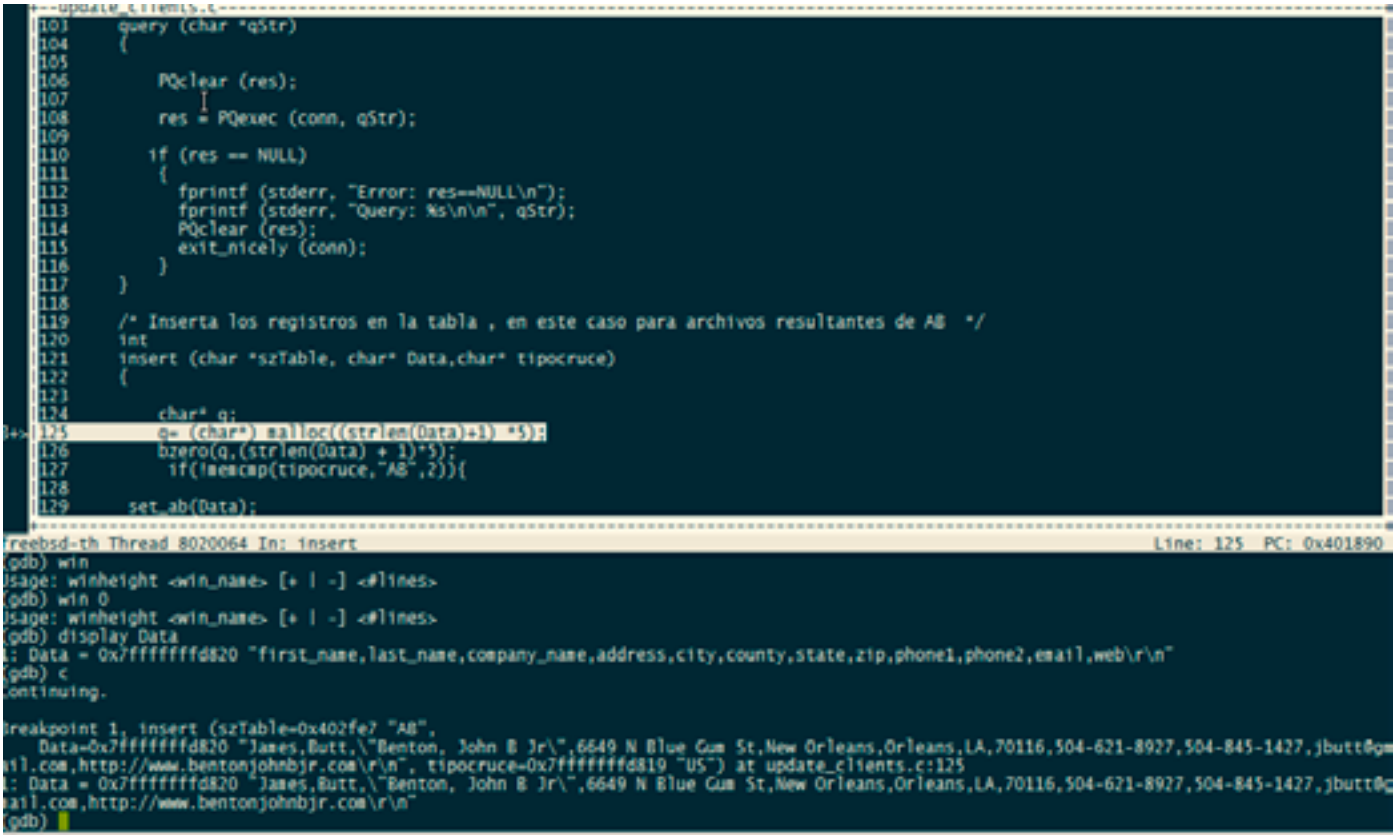


Figure 11. *Running gdb using display command*

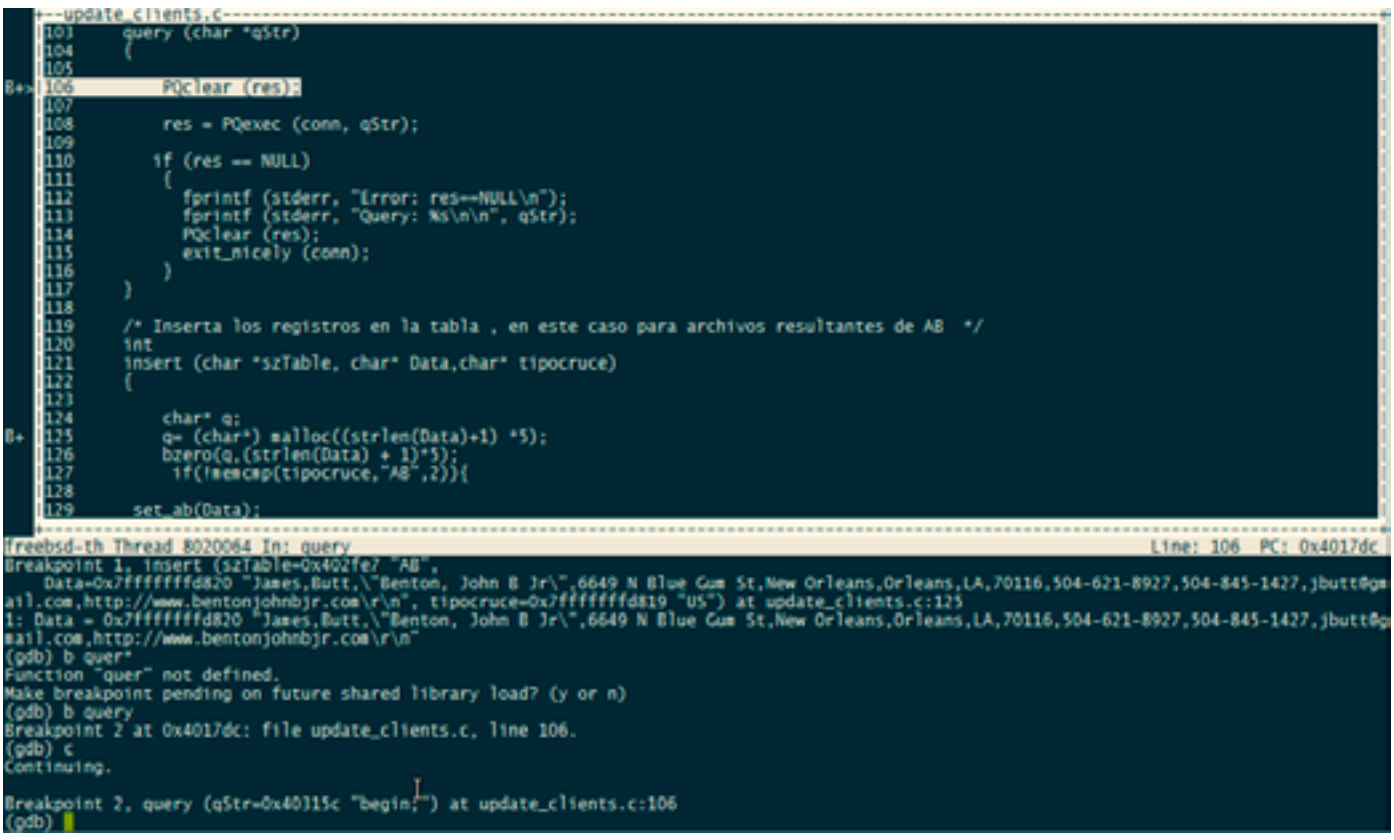


Figure 12. *Running gdb stopping at a breakpoint*

the libpq library (*http://www.postgresql.org/docs/9.1/static/libpq-exec.html*). It seems not enough to check for NULL. To really know what the database tells us about the result of each transaction, we will use the following functions:

PQresultErrorMessage
Returns the error message associated with the query, or an empty string if there was no error. const char *PQresultErrorMessage(PGresult *res);

PQresultStatus
Returns the result status of the query. PQresult- Status can return one of the following values:

```
PGRES_EMPTY_QUERY,
```

```
void
query (char *qStr)
{

   PQclear (res);

   res = PQexec (conn, qStr);
if( (PQresultStatus(res) != PGRES_COMMAND_OK) )
   {
     fprintf (stderr, "Error: res==NULL\n");
     fprintf (stderr, "Query: %s\n\n", qStr);
     fprintf(stderr, "error <%s>", PQresultErrorMessage(res));
     PQclear (res);
     exit_nicely (conn);
   }
}
```

Figure 13. *Code snipet for the Query function*

```
PGRES_COMMAND_OK,  /* the query was a command returning
  no data */
PGRES_TUPLES_OK,  /* the query successfully returned
  tuples */
PGRES_COPY_OUT,
PGRES_COPY_IN,
PGRES_BAD_RESPONSE, /* an unexpected response was
  received */
PGRES_NONFATAL_ERROR,
PGRES_FATAL_ERROR
```

Let's run it again, the result will be as follows: see Figure 14. *Where did that come from*?, looks like somebody implemented a function that tries to insert a hash but never worked. By looking at the file, when it was created they never took out the header from the csv file as shown: see Figure 15.

Let's remove that in the file, and try again. Now, we have another problem according to the table of data types shown as follows: see Figure 16.

It makes no sense that "LA" is being considered as if it were a zip code (see Figure 17).

Then, I'll set a breakpoint in the *extract_field* function when it tries to extract the value for the 7th field. I don't want to wait for all fields to be processed as just set a condition in the breakpoint as follows (see Figure 18):

```
int modulo_11(char* szrut)
{
cq
cneira@trueos:~/workshop/Intro/source % sudo ./update_clients ../data/us.csv
Password:
digest es :72c8a1214e9cb1e770213b206a07d475b82b30b1
Error: res==NULL
Query: insert into cruces (id) values( 'r□NCCCp!; j□u□+0□');

error <ERROR:  invalid byte sequence for encoding "UTF8": 0x9c
HINT:  This error can also happen if the byte sequence does not match the encoding expected by the server, which is controlled by "client_en
coding".
cneira@trueos:~/workshop/Intro/source %
```

Figure 14. *Running gdb again*

```
/* inserta los registros en la tabla , en este caso para archivos resultantes de AD  */
int
insert (char *szTable, char* Data,char* tipocruce)
update_clients.c: 444 lines, 12819 characters.
cneira@trueos:~/workshop/Intro/source % sudo ./update_clients ../data/us.csv
Error: res==NULL
Query: insert into "us"  (first_name,last_name, company_name,address,city,county,state, zip,phone1,phone2,email,web)    values ( 'first_name
', 'last_name', 'company_name', 'address' ,'city' ,'county' ,'state','zip','phone1','phone2','email','web
);

error <ERROR:  invalid input syntax for integer: "zip"
LINE 1: ...apany_name', 'address' ,'city' ,'county' ,'state','zip','pho...
                                                             ^
cneira@trueos:~/workshop/Intro/source %
```

Figure 15. *Running gdb query returns error*

```
Query: insert into "us"  (first_name,last_name, company_name,address,city,county,state, zip,phone1,phone2,email,web)    values ( 'James', 'B
utt', 'Benton', ' John B Jr"' ,'6649 N Blue Gum St' ,'New Orleans' ,'Orleans','LA','70116','504-621-8927','504-845-1427','jbutt@gmail.com')
;

error <ERROR:  invalid input syntax for integer: "LA"
LINE 1: ...' ,'6649 N Blue Gum St' ,'New Orleans' ,'Orleans','LA','7011...
                                                             ^
cneira@trueos:~/workshop/Intro/source %
```

Figure 16. *Running gdb SQL error*

COLUMN_NAME	TYPE_N...	IS_NULLA...	DECIMAL_DIGI...	COLUMN_...	COLUMN...	REMA...	DATA...
first_name	varchar	NO	0	2147483647	<null>	<null>	12
last_name	varchar	NO	0	2147483647	<null>	<null>	12
company_name	varchar	NO	0	2147483647	<null>	<null>	12
address	varchar	NO	0	2147483647	<null>	<null>	12
city	varchar	NO	0	2147483647	<null>	<null>	12
county	varchar	NO	0	2147483647	<null>	<null>	12
state	varchar	NO	0	2147483647	<null>	<null>	12
zip	int8	NO	0	19	<null>	<null>	-5
phone1	varchar	NO	0	2147483647	<null>	<null>	12
phone2	varchar	NO	0	2147483647	<null>	<null>	12
email	varchar	NO	0	2147483647	<null>	<null>	12
web	varchar	NO	0	2147483647	<null>	<null>	12

Figure 17. *us table data types*

```
(gdb) r ../data/us.csv
Starting program: /usr/home/cneira/workshop/Intro/source/update_clients ../data/us.csv
[New LWP 100538]
[New Thread 802006400 (LWP 100538/update_clients)]
[Switching to Thread 802006400 (LWP 100538/update_clients)]

Breakpoint 1, extract_field (
    string=0x7fffffffd820 "James,Butt,\"Benton, John B Jr\",6649 N Blue Gum St,New Orleans,Orleans,LA,70116,504-621-8927,504-845-1427,jbutt@
gmail.com,http://www.bentonjohnbjr.com\r\n", nfield=1, fieldsize=0, separator=0x403391 ",") at update_clients.c:242
warning: Source file is more recent than executable.

242
Current language:  auto; currently minimal
(gdb) break if nfield == 7
Note: breakpoint 1 also set at pc 0x4026b6.
Breakpoint 2 at 0x4026b6: file update_clients.c, line 242.
(gdb) i b
Num Type           Disp Enb Address            What
1   breakpoint     keep y   0x00000000004026b6 in extract_field at update_clients.c:242
        breakpoint already hit 1 time
2   breakpoint     keep y   0x00000000004026b6 in extract_field at update_clients.c:242
        stop only if nfield == 7
(gdb) d b 1
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) n
Program not restarted.
(gdb) c
Continuing.
```

Figure 18. *extract_field*

```
(gdb) n
256             field = strtok_r (NULL, separator, &brk);
3: field = 0x8020653ff "6649 N Blue Gum St"
2: nfield = 7
1: nfield = 7
(gdb) n
258         }
3: field = 0x8020653ff "6649 N Blue Gum St"
2: nfield = 7
1: nfield = 7
(gdb) n
256             field = strtok_r (NULL, separator, &brk);
3: field = 0x802065412 "New Orleans"
2: nfield = 7
1: nfield = 7
(gdb) n
258         }
3: field = 0x802065412 "New Orleans"
2: nfield = 7
1: nfield = 7
(gdb) n
256             field = strtok_r (NULL, separator, &brk);
3: field = 0x80206541e "Orleans"
2: nfield = 7
1: nfield = 7
(gdb) n
261
3: field = 0x80206541e "Orleans"
2: nfield = 7
1: nfield = 7
(gdb) n
262     int modulo_11(char* szrut)
3: field = 0x80206541e "Orleans"
2: nfield = 7
1: nfield = 7
(gdb)
insert (szTable=0x4030b7 "AB",
    Data=0x7fffffffd820 "James,Butt,\"Benton, John B Jr\",6649 N Blue Gum St,New Orleans,Orleans,LA,70116,504-621-8927,504-845-1427,jbutt@gm
ail.com,http://www.bentonjohnbjr.com\r\n", tipocruce=0x7fffffffd819 "US") at update_clients.c:181
181             extract_field(Data,10,0,","),
3: field = 0x80206541e "Orleans"
(gdb) n
```

Figure 19. *checking the values for the local variables*

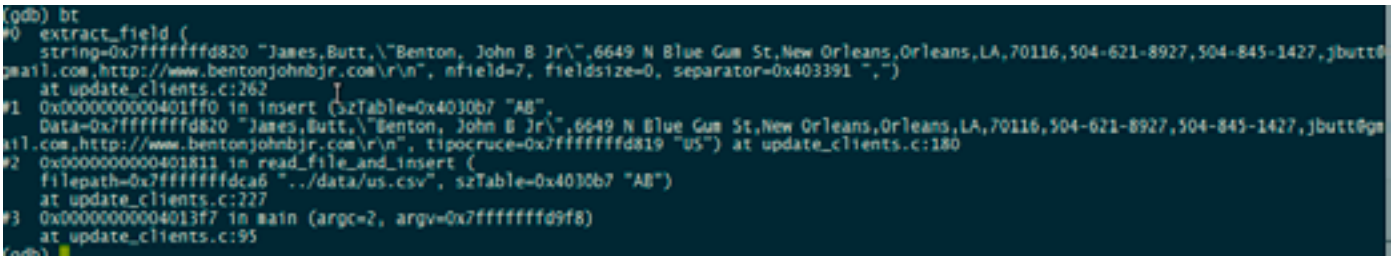


Figure 20. *extract_field back trace*

```
(gdb)  b if nfield == 7
```

I'll just type c (short form of continue) to resume the program execution, I'll go over every instruction from the program and check the values for the local variables. In this case I have stopped at the extract_field function as I'm checking why the "LA" value is being considered as a zip code (integer, see Figure 19). And to display the variable value I'm interested in during this debug session, just by typing:

```
 display <variable> will do, for example in this case:
(gdb) display  field
(gdb) display nfield
```

These variables that live within the scope of the extract_field function. If I don't want to display one of the values anymore I just have to type: `undisplay <variable>`.

This is an interesting extraction of the field at position 7 and the value is "Orleans", but according to the data, the value should be "LA". So, w*hat is the problem?* it seems that the extract_field has a bug, this thing is off by a field. I'll check the *backtrace* to remember how I got to this point. Typing "bt" shows me the backtrace.

A backtrace is a summary of how your program got where it is. It shows one line per frame, for many frames, starting with the currently executing frame (frame zero), followed by its caller (frame one), and on up the stack (see Figure 20).

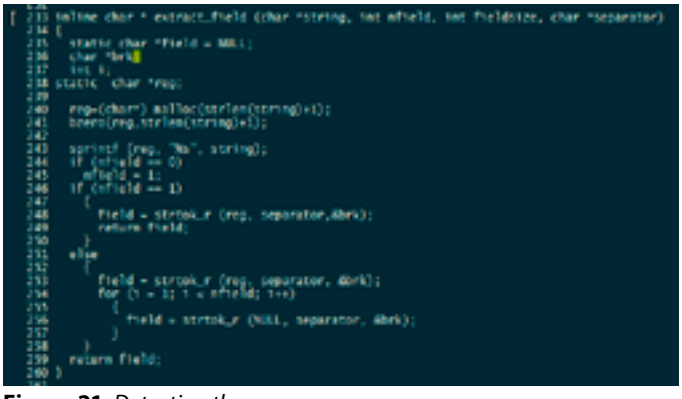


Figure 21. *Detecting the error*

There it is, the whole string being tokenize by the extract field function is "James,Butt....", as we see in frame 0, the interesting part is the argument separator = "," and the third field of the string: \"Benton, John B Jr\".

The bug there is a semicolon in the third field causing the "LA" value to be considered as a zip code.

Let's fix the code at runtime. I'll set a breakpoint at line 242 and replace the string \"Benton, John B Jr\" with \"Benton John B Jr"\ and see how it goes (Figure 21).

And to set a breakpoint at a specific line of code, use the following commands:

```
as in (gdb) b  <source.c>:<line number>
```

```
(gdb) b update_clientes.c:242.
```

Now let's run this again: see Figure 22.

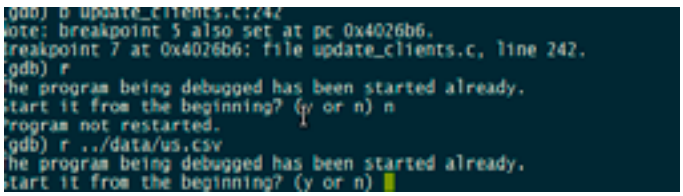


Figure 22. *Running again after correcting the error*

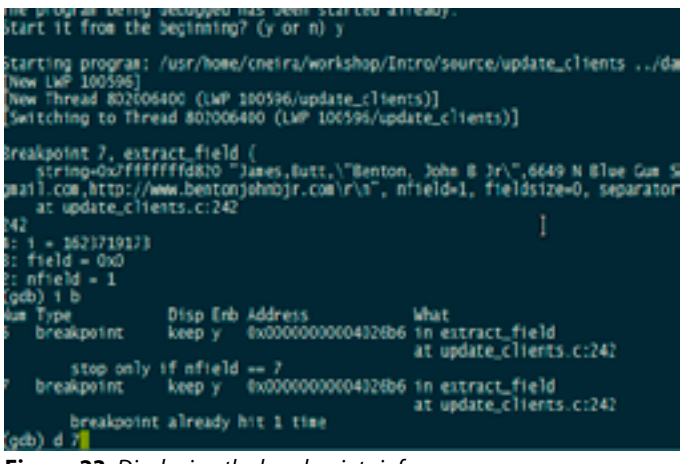


Figure 23. *Displaying the breakpoints info*

I had some breakpoints set, so I'll delete them by typing d and the breakpoint number, to know which breakpoints we need to type as follows:

```
(gdb)  i  b
```

That is the short form of breakpoints information: see Figure 23.

Now, I'll replace the actual string value with the one I want using the following command:

```
(gdb)  set <variable>  <value>
```

I have changed the value, but still it is failing. That is because I picked the wrong breakpoint to change the data. We should have set a breakpoint at line 121 where the

```
Breakpoint 5, extract_field (
    string=0x7fffffffd820 "James,Butt,\"Benton, John B Jr\",6649 N Blue Gum St,New Orleans,Orleans,LA,70116,504-621-8927,504-845-1427,jbutt@
gmail.com,http://www.bentonjohnbjr.com\r\n", nfield=7, fieldsize=0, separator=0x403391 ",")
    at update_clients.c:242
242
$1 = 6
$2 = field = 0x802065372 "New Orleans"
$3 = nfield = 7
(gdb) d 3 2
No breakpoint number 3.
No breakpoint number 2.
(gdb) ud 3 2
Undefined command: "ud".  Try "help".
(gdb) undisplay 3 2
(gdb) p string
$3 = 0x7fffffffd820 "James,Butt,\"Benton, John B Jr\",6649 N Blue Gum St,New Orleans,Orleans,LA,70116,504-621-8927,504-845-1427,jbutt@gmail.
com,http://www.bentonjohnbjr.com\r\n"
(gdb) display string
4: string = 0x7fffffffd820 "James,Butt,\"Benton, John B Jr\",6649 N Blue Gum St,New Orleans,Orleans,LA,70116,504-621-8927,504-845-1427,jbutt
@gmail.com,http://www.bentonjohnbjr.com\r\n"
(gdb) set string="James,Butt,\"Benton, John B Jr\",6649 N Blue Gum St,New Orleans,Orleans,LA,70116,504-621-8927,504-845-1427,jbutt@gmail.com
,http://www.bentonjohnbjr.com\r\n"
(gdb) set string="James,Butt,\"Benton John B Jr\",6649 N Blue Gum St,New Orleans,Orleans,LA,70116,504-621-8927,504-845-1427,jbutt@gmail.com,
http://www.bentonjohnbjr.com\r\n"
(gdb) show values 10
(gdb) c
Continuing.
Error: res==NULL
Query: insert into "us"  (first_name,last_name, company_name,address,city,county,state, zip,phone1,phone2,email,web)    values ( 'James', 'B
utt', '"Benton', ' John B Jr"' ,'6649 N Blue Gum St' ,'New Orleans' ,'LA','LA','70116','504-621-8927','504-845-1427','jbutt@gmail.com');

Error <ERROR:  invalid input syntax for integer: "LA"
LINE 1: ...B Jr"' ,'6649 N Blue Gum St' ,'New Orleans' ,'LA','LA','7011...
                                                             ^
```

Figure 24. *Replacing actual string value*

References

- "31.3. Command Execution Functions." PostgreSQL: Documentation: 9.1: Command Execution Functions. Web. 08 Aug. 2014. *http://www.postgresql.org/docs/9.1/static/libpq-exec.html*.
- "5.5. Using the Ports Collection." 5.5. Using the Ports Collection. Web. 06 Aug. 2014. *http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/ports-using.html*.
- "All Debuggers – GNU Compiler Collection (GCC) Internals." All Debuggers – GNU Compiler Collection (GCC) Internals. Web. 06 Aug. 2014. *https://gcc.gnu.org/onlinedocs/gcc-4.7.4/gccint/All-Debuggers.html*.
- "Clang 3.6 Documentation." Clang Compiler User's Manual –. Web. 06 Aug. 2014. *http://clang.llvm.org/docs/UsersManual.html#cmdoption-g*.
- "Debugging with GDB – Backtrace." Debugging with GDB – Backtrace. Web. 10 Aug. 2014. *https://ftp.gnu.org/old-gnu/Manuals/gdb-5.1.1/html_node/gdb_42.html*.
- "Detailed Installation Guides." – PostgreSQL Wiki. Web. 10 Aug. 2014. https://wiki.postgresql.org/wiki/Detailed_installation_guides#FreeBSD.

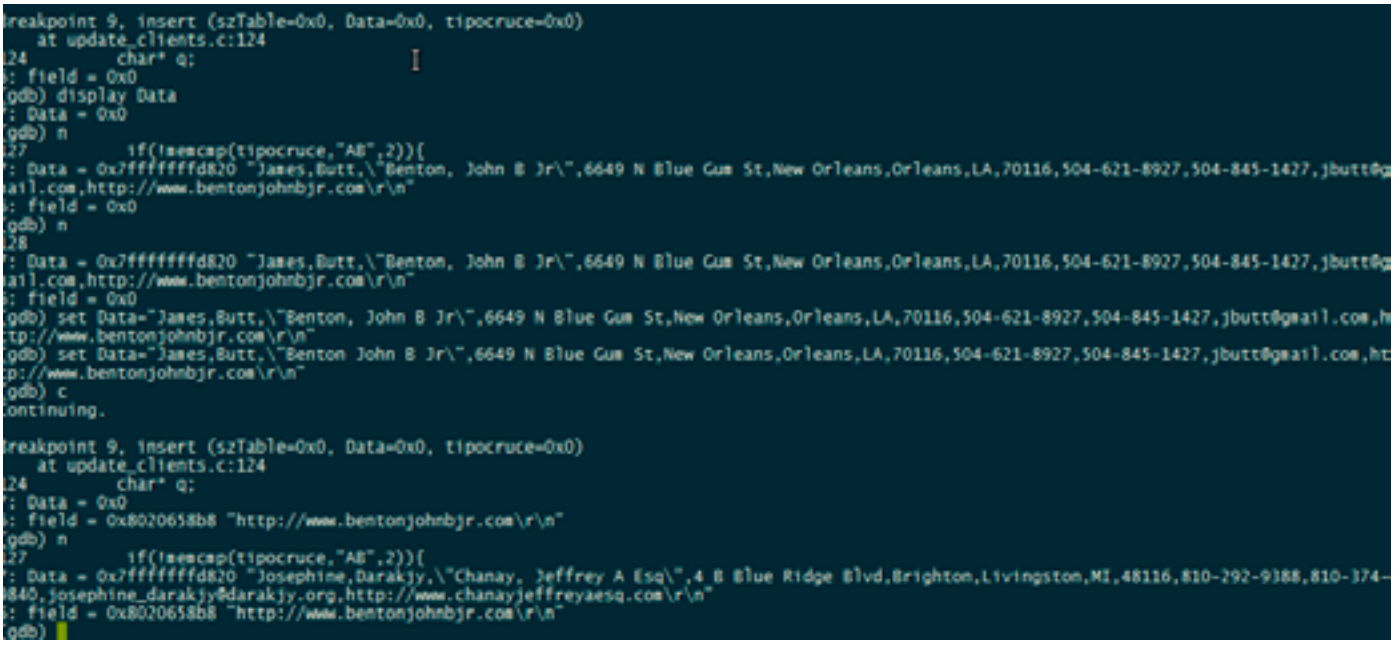


Figure 25. *Re-setting the breakpoint*

string is being passed as a parameter to the function that calls this subroutine. *Let's fix this and go home!*

It worked for the record, we have modified (no SQL error at insert), but here comes another register with the same issue as the one we have corrected. At least, we know the fix we applied at the runtime made it work. So now, it is just a matter of modifying the data and reprocessing the data at this point.

This all seems easy at this point but...!! what if we are not allowed to modify the source code to make the necessary changes for this to work out? What if we don't have the source code to debug?

For that situation, we could use some library interposers and add a wrapper around the function that is causing us trouble. We could also use gdb to debug even if we don't have the source code available.

All these topics will be in another tutorial continuing this tutorial series.

Conclusions

The GNU debugger (*gdb*) is a really powerful tool that gives you an edge advantage when troubleshooting an application. As a developer, you should be proficient using it and it will serve you well.

CARLOS ANTONIO NEIRA BUSTOS

Carlos Neira has worked several years as a C/C++ developer and kernel porting and debugging enterprise legacy applications. He is currently employed as a C developer under Z/OS, debugging and troubleshooting legacy applications for a global financial company. Also he is engaged in independent research on affective computing. In his free time he contributes to the PC-BSD project and enjoys metal detecting.

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

Deploying an Office Server In FreeBSD, With File Sharing and E-mail

FreeBSD is a modern and capable operating system (OS) which can be both robust and easily manageable if used in an office or a workgroup server environment. It supports the whole range of cutting edge Open-Source technologies, which makes using it a completely pleasant and well-featured experience. The latest FreeBSD release, 10.0, delivers a bunch of improvements which increase both the performance and usability of this fine operating system.

We have prepared a workshop which aims to teach users with minimal knowledge of FreeBSD in a step by step guide how to install and configure a usable office server from scratch. This server will be intended to provide office workers or collaborators with a modern central point used to share data, which they can trust and rely on in their daily work.

Its main features will be document sharing, collaboration and E-mail. The various tasks of the server will be handled by different standard Open-Source tools as we will use Samba and the built-in FreeBSD Network File System (NFS) servers for serving files in the local network, ownCloud for sharing files across the public Internet, Apache and PHP for serving web applications, Postfix for the SMTP server and finally Dovecot for the E-mail server. We will also add a webmail interface to our server using RoundCube and we will finally protect our server by using tools such as ipfw and sshit.

First Step: Installing FreeBSD

Starting with the previous major release of FreeBSD (9.0), the Operating System can be installed with a new installer which supports more of its new technologies

Figure 1. *The new FreeBSD installer allows the use of current technologies while being user-friendly enough for new users*

than its predecessor. At the same time, it offers more manual configuration options for expert users.

Since the tutorial is about deploying a server, the hardware configuration on which the system will be installed is assumed to contain two *Hard Drives* (HD), which will be mirrored in software using FreeBSD's GEOM_MIRROR facility. The usage of two drives for the server is a common compromise between reliability and cost, and the same can be said about using Software RAID instead of a Hardware RAID controller. For a small office server, generic desktop drives of suitable capacity (e.g. 1.5 TB or 2 TB) are a fine choice, but "enterprise-class" drives are not that expensive and can be worth searching, if only because they usually have more mature firmware and disabled on-drive write caching.

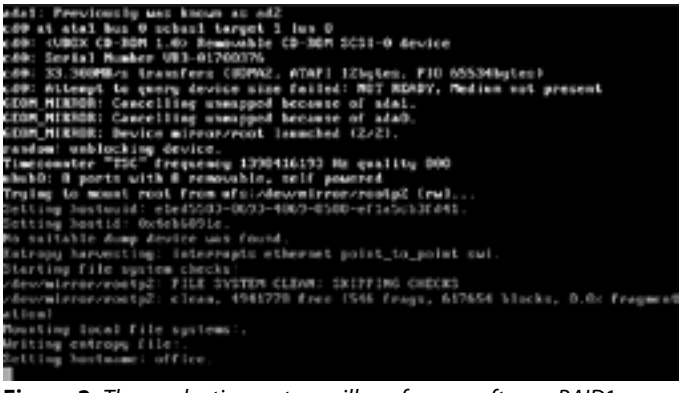


Figure 2. *The production system will run from a software RAID1 volume powered by GEOM_MIRROR*

This initial step of the process is the only one which requires a small detour in the default installation process, in which the required kernel module for GEOM_MIRROR will be loaded and the RAID volume is created. The rest of the installation process is smart enough to recognize such manually created devices and can use them to create partitions and file systems.

Our setup will use the default FreeBSD file system, which in the most recent version, is a variant of the UFS2 file system with soft-updates-journaling enabled. For those who are experienced with Linux, the characteristics of this file system (very) loosely correspond to ext4 with the data=writeback option enabled. An alternative file system for FreeBSD could be ZFS, which is one of the options presented in the installer but marked as experimental, and for this, it will be covered briefly.

The New Package Manager

The traditional way of installing software on FreeBSD (and many other Unix-like operating systems) is by compiling a code. The BSD systems have evolved infrastructure (the ports collections) which makes this much easier and offers somewhat advanced features such as dependency tracking, but practical daily use of ports still requires expert knowledge that is not required by the more streamlined Linux systems. Though the ports can be used (and regularly are used) to build binary packages, these packages were until recently both much less flexible and rarely built, which made them an inferior choice compared to ports.

Finally, the recent version of FreeBSD (V10.0) brings in a modern binary package manager called "pkg", with a new infrastructure and a new approach to binary packages. They are no longer second-class citizens of the FreeBSD user land but a fully supported and maintained way of maintaining software, for the most part that is removed from the quirks of using ports. The package manager is steadily improving and can now deal with most of the situations which arise in daily use (such as dependency issues), and the default package repository for FreeBSD contains almost all software available in ports. Being prebuilt with default options, the binary packages are still less flexible than ports, but the strategies to reduce the difference in flexibilities are actively under development.

Though the new package manager is called "pkg" (also called "pkg-ng" but that name is now obsolete), it does not share code with other software with the same name, and most notably that from Solaris. Unusually for a BSD, the package manager is itself NOT a part of the base system, but is installed seamlessly on first use.

Figure 3. *As the BSD systems are traditionally divided into the "base" system and the third party applications, the installer only needs to install the "base" and "kernel" files*

Local File Sharing with Samba

Samba is the famous Open-source project which brings compatibility with *Microsoft's* file sharing technologies to non-Windows operating systems. It is an important project which receives regular updates and is maintained to be compatible with the latest Windows variants. The latest version of Samba can act as Active Directory Domain

Controllers, which expands their capabilities and opens up new use cases. Samba under FreeBSD works mostly out of the box, but requires some moderate tuning to be as high-performing as the users expect it to be.

Local File Sharing with NFS

The *Network File System* (NFS) is the preferred local file sharing protocol between Unix-like systems, mostly due to its ubiquitous presence in such system and the relative simplicity of its operations. Consequently, it is worth using only between such systems, as it's usually poorly suitable for truly heterogeneous environments. FreeBSD's native NFS server and client are well supported and fairly high-performing, and require minimal configuration and no third party software to get running.

Apache and PHP For Web Applications

The combination of the Apache web server and the PHP programming language is the most common web application infrastructure on the Internet. The large volume of applications written in PHP and the relative simplicity of their setup / installation make it attractive for the office server. Indeed, all other web applications which will be covered by this tutorial are written for PHP and will be powered by this very setup. A very important aspect of running a web server today is SSL / TLS, a protocol which provides end-to-end encryption used in HTTPS. The part of the workshop dealing with Apache will also cover creating and submitting an SSL certificate request, as well as its installation.

File Sharing and Collaboration over The Internet With ownCloud

While Samba and NFS are perfectly suitable for sharing files in the local network (e.g. within an office or in a company), they were not created for sharing files over the wider Internet. They lack the flexibility and security properties needed in the global environment with unknown users and unreliable connectivity. The recently prominent Open-source project "ownCloud" will be used in our configuration to provide file sharing and collaboration across the Internet. It is a powerful tool which consists of several applications, and file sharing is just one of the options that it supports. Among its basic features, it supports shared contacts and calendar, and a Dropbox-like desktop file synchronization utility, but it also supports adding third-party applications and extensions which greatly increase its usability.

E-mail Servers With Postfix and Dovecot

The E-mail system used today relies on two types of protocols: for routing E-mail to and between E-mail servers, and for retrieving E-mail from those servers. The protocol of the first type is the Simple Mail Transfer Protocol (SMTP), implemented (among other products) by Postfix. There are several protocols of the second type, but the most feature-rich and the most popular today is IMAP, implemented (also, among other products), by Dovecot.

Figure 4. *OwnCloud is a web application with several parts, among which are a Dropbox-like file synchronization service and a shared calendar*

An important part of running an e-mail server is spam protection. This is a topic which can get very complex very quickly, but the workshop will guide through basic anti-spam measures which include acceptance rules for the SMTP server and the SpamAssassin software for active e-mail scanning.

WebMail with RoundCube

E-mail is traditionally accessed by desktop software (e.g. Thunderbird, Windows Live Mail, eM Client or Zimbra Desktop) but using a web-based application is becoming increasingly convenient because it doesn't require additional software installation and the web can be accessed through corporate and hotel firewalls.

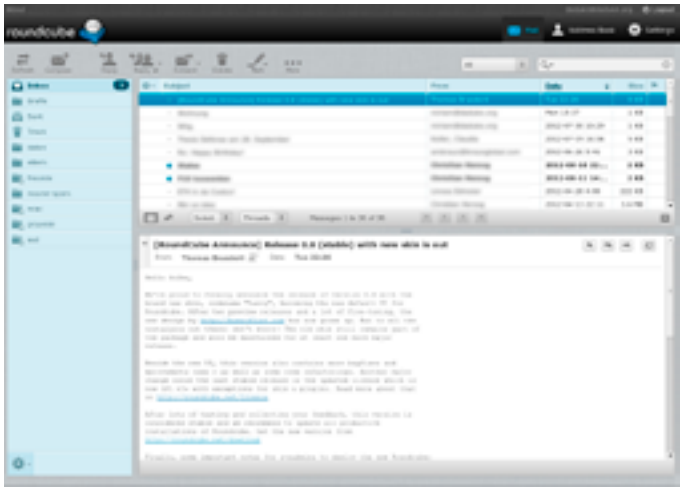


Figure 5. *RoundCube is a web GUI for IMAP file servers with a familiar and simple interface and powerful options and plugins*

RoundCube is a web application written in PHP which can act as an IMAP client and present all the E-mail available on the server in a modern and pretty web-based user interface. RoundCube is a web application written in PHP which can act as an IMAP client and present all the e-mail available on the server in a modern and pretty web-based user interface.

Protecting The Server With ipfw and sshit

As the server in this tutorial contains services intended to be used over the Internet, appropriate effort needs to be undertaken to ensure both the server and its services are resilient to common attacks which are a matter-of-course on the open Internet.

FreeBSD's default firewall is ipfw, with an easy and straightforward syntax and optional stateful packet inspection. A good (and always welcome) addition to it is the sshit package which blocks brute-force attacks over ssh.

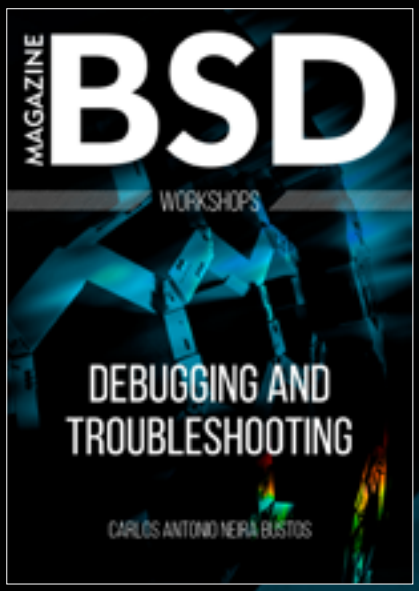
Final Thoughts

The goal of this tutorial is to teach users with moderate experience in any Unix-like system to install and deploy a quality office server with common applications and services. To ensure this, the tutorials of the workshop will cover not only how something is done but also why it's done. And this will also be reflected in the final test.

IVAN VORAS

Ivan Voras is a FreeBSD developer and a long-time user, starting with FreeBSD 4.3 and throughout all its versions' history. On the practical side, he is a researcher, system administrator and a developer, as the opportunity presents itself, with a wide range of experience from hardware hacking to cloud computing. He is currently employed at the University of Zagreb, Faculty of Electrical Engineering and Computing and currently lives in Zagreb, Croatia. You can reach him through: English Blog: http://ivoras.net/blog | Croatian Blog: http://hrblog.ivoras.net/ | Google+: https://plus.google.com/.

DEPLOYING AN OFFICE / WORKGROUP SERVER ON FREEBSD, WITH E-MAIL AND FILE SHARING



WORKSHOP ABSTRACT

USING FREEBSD AS A SERVER FOR COMMON OFFICE DAILY TASKS IS EASY AND AN APPROACHABLE TASK EVEN FOR USERS HAVING NO EXTENSIVE KNOWLEDGE OF ITS INTERNALS. IN THIS WORKSHOP, YOU WILL LEARN HOW TO BRING UP A FUNCTIONING SERVER FOR A SMALL OFFICE OR A WORKGROUP, WHICH INCLUDES A SMALL WEB SERVER, E-MAIL WITH POSTFIX FOR SMTP, DOVECOT FOR IMAP AND ROUNDCUBE FOR THE WEBMAIL USER INTERFACE, SAMBA FOR LOCAL FILE SHARING (CIFS / WINDOWS NETWORKING / NETWORK NEIGHBOURHOOD) AND PYDIO FOR REMOTE FILE ACCESS OVER THE WEB.

WORKSHOP REQUIREMENTS

1. BASIC KNOWLEDGE OF UNIX/LINUX SYSTEMS
2. BASIC FAMILIARITY WITH COMMAND-LINE OPERATIONS
3. SYSTEM (POSSIBLY A VIRTUAL MACHINE) ON WHICH THE STUDENT WILL PERFORM THE REQUIRED TASKS

WORKSHOP AGENDA

MODULE 1 – INTRODUCTION AND SERVER SETUP

TUTORIAL #1 – INTRODUCTION, DESCRIPTION OF WHAT WE ARE GOING TO ACCOMPLISH, OVERVIEW OF TECHNOLOGIES INVOLVED, HOW AND WHY TO USE THEM.

TUTORIAL #2 – DESCRIPTION OF VARIOUS PRODUCTS (PACKAGES) AND PROTOCOLS WHICH WILL BE INVOLVED IN SETTING UP OUR SERVER.

TUTORIAL #3 – INSTALLING A FREEBSD 10 SYSTEM, ADVICE ON CHOOSING HARDWARE, DISK AND NETWORK SETTINGS, AND COMMONLY USED UTILITY PACKAGES.

MODULE 2 – INSTALLING A WEB SERVER AND A FILE SHARING WEB APPLICATION

TUTORIAL #4 – CONFIGURING A SIMPLE WEB SERVER WITH APACHE AND PHP

TUTORIAL #5 – CONFIGURING SSL / TLS: DESCRIPTION OF HOW IT WORKS, CREATING A CERTIFICATE REQUEST, SUBMITTING IT TO CACERT.ORG AND INSTALLING THE RESULTING CERTIFICATE

TUTORIAL #6 – CONFIGURING A FILE-SHARING WEB APPLICATION USING PYDIO

MODULE 3 – INSTALLING THE E-MAIL SERVERS AND THE WEBMAIL INTERFACE

TUTORIAL #7 – CONFIGURING AN E-MAIL SERVER USING POSTFIX, INCLUDING ANTI-SPAM OPTIONS.

TUTORIAL #8 – CONFIGURING AN IMAP SERVER USING DOVECOT.

TUTORIAL #9 – CONFIGURING A WEBMAIL INTERFACE TO THE IMAP SERVER USING ROUNDCUBE.

MODULE 4 – LOCAL FILE SHARING AND FIREWALL

TUTORIAL #10 – CONFIGURING WINDOWS FILE SHARING WITH SAMBA

TUTORIAL #11 – CONFIGURING UNIX FILE SHARING WITH NFS

TUTORIAL #12 – PROTECTING YOUR SERVER WITH A NETWORK FIREWALL USING IPFW

INSTRUCTOR BIO

IVAN VORAS IS A FREEBSD DEVELOPER AND A LONG-TIME USER, STARTING WITH FREEBSD 4.3 AND THROUGHOUT ALL ITS VERSIONS' HISTORY. ON THE PRACTICAL SIDE, HE IS A RESEARCHER, SYSTEM ADMINISTRATOR AND A DEVELOPER, AS THE OPPORTUNITY PRESENTS ITSELF, WITH A WIDE RANGE OF EXPERIENCE FROM HARDWARE HACKING TO CLOUD COMPUTING. HE IS CURRENTLY EMPLOYED AT THE UNIVERSITY OF ZAGREB, FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING AND CURRENTLY LIVES IN ZAGREB, CROATIA.
YOU CAN REACH HIM THROUGH:
ENGLISH BLOG: HTTP://IVORAS.NET/BLOG
CROATIAN BLOG: HTTP://HRBLOG.IVORAS.NET/
GOOGLE+: HTTPS://PLUS.GOOGLE.COM/+IVANVORAS.

IF YOU WANT TO PARTICIPATE IN OUR BSD ONLINE WORKSHOPS, PLEASE CONTACT UROS DRNOVSEK AT UROS.DRNOVSEK@BSDMAG.ORG DIRECTLY.

Return Oriented Programming

Since 1988, the Morris Worm stack overflow has been a nightmare for developers. Several countermeasures have been created to avoid this kind of attack. Compilers are pioneers in developing such techniques.

Sadly, few programmers know very much about compilers' options as they usually compile programs with inherited procedures. For instance, the very well known GCC compiler has a stack protection with the *fstack-protector option* [1].

In the middle of the past decade, manufacturers introduced the *No-eXecute* (NX) bit which prevents the execution of code beyond the text area of a program. When this bit is ON, the processor sends a signal to the *Operating System* (OS). In addition, it is also necessary for the Operating System to be instructed to stop the code execution. In Windows, this is achieved by activating the Data Execution Prevention.

Readers must be aware that the *NX bit does not prevent stack overflow* and only prevents the execution of injected code. So, if you are able to exploit such a vulnerability, you are completely free to write anything you like in the stack. However, a clever hacker may think…. "Of course, I can't execute code but I can alter the normal flow of execution, making the program go to another address by means of overwriting the return address located in the stack".

As a concept of proof, we will work with this simple program:

```
#include <ctype.h>
#include <stdio.h>
#include <string.h>
int tabla[5] = {91, 92, 93, 94, 95};
{
FILE *fd;
int in1,in2;
int arr[20];
char var[20];

 if (argc !=2){
  printf(mensaje0,*argv);
  return -1;
 }
 fd = fopen(argv[1],"r");
 if(fd == NULL)
 {
  fprintf(stderr,mensaje1);
  return -2;
 }
 memset(var,7,sizeof(var));
 memset(arr,6,20*sizeof(int));
 while(fgets(var,20,fd))
 {
  in1 = atoll(var);
  fgets(var,20,fd);
  in2 = atoll(var);
  /* fill array */
  arr[in1]=in2;
  //printf("%d - %d\n", arr[in1], tabla[in1]);
  if (arr[in1] != tabla[in1])
  {
  printf("Sorry values are no correct!\n");
  return ;
  }
  printf("Correct". The process follows\n");
  printf("Your are in the core of the program\n");
```

```
staff@caucaso /practicas
$ cat ropok.txt
2
93
staff@caucaso /practicas
$ rop2.exe ropok.txt
Data are correct entering
In this point you have entered in program flow accesible only when you have intr
oduced correct values in file parameter.
```

Figure 1.

```
staff@caucaso /practicas
$ cat ropnotok.txt
2
95
staff@caucaso /practicas
$ rop2.exe ropnotok.txt
La coordenada no es correcta
Data are not correct exiting
```

Figure 2.

```
  return;
 }
}
```

Code Logic

The program reads a file with 2 lines; each line contains a number (in1 & in2), in1 is used as the index. If the value contained in the cell table[in1] is equal to in2, then the

```
004017D5  .  8B4424 78     MOV EAX,DWORD PTR [ESP+78]
004017D9  .  8B0485 004040 MOV EAX,DWORD PTR [EAX*4+404000]
004017E0  .  39C2          CMP EDX,EAX
004017E2  .v 74 0E         JE SHORT rop20.004017F2
004017E4  .  C70424 665040 MOV DWORD PTR [ESP],rop20.00405066   ASCII "Data are not co
004017EB  .  E8 E01F0000   CALL <JMP.&msvcrt.puts>              puts
004017F0  .v EB 19         JMP SHORT rop20.0040180B
004017F2  >  C70424 835040 MOV DWORD PTR [ESP],rop20.00405083   ASCII "Data are corre
004017F9  .  E8 D21F0000   CALL <JMP.&msvcrt.puts>              puts
004017FE  .  C70424 204040 MOV DWORD PTR [ESP],rop20.00404020   ASCII "In this point y
00401805  .  E8 9E1F0000   CALL <JMP.&msvcrt.printf>            printf
0040180A  .  90            NOP
0040180B  >  C9            LEAVE
0040180C  L. C3            RET
```

Figure 3.

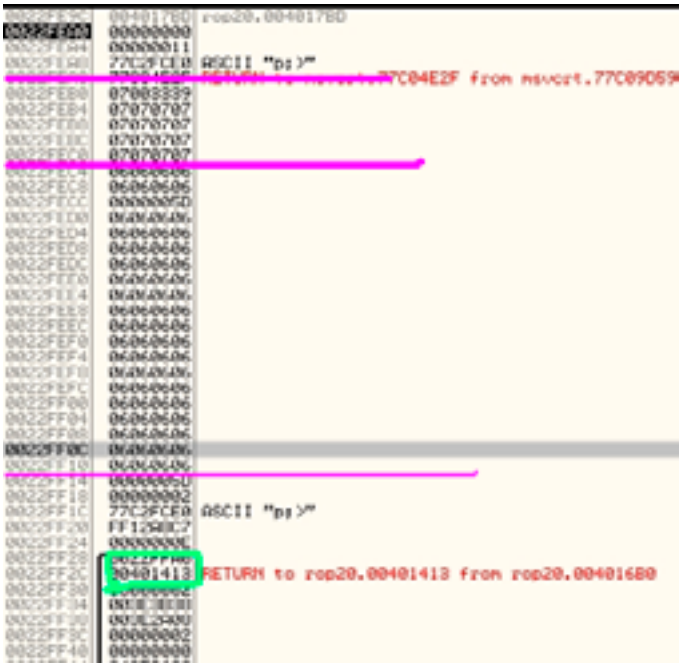


Figure 4.

process is OK and will continue; otherwise, the process terminates. In a real environment, the table will be out of the program, even encrypted or secured with another security measure; but for us, this is not relevant because the only matter we must deal with is the return address.

Readers may wonder at these odd initializations:

```
memset(var,7,sizeof(var));
memset(arr,6,20*sizeof(int));
```

They are only just a trick to make these values more visible in the stack area. And this is what happens when parameter values contained in the file are: 2 (in the first line) and 93 (in the second line): see Figure 1.

And as shown in the next figure, this is what will happen when the parameter file contains incorrect values: 2, 95: see Figure 2.

Now, we start the program under Ollydbg [2] and we should make a breakpoint when jumping depending on the values in the parameter file. When parameters are set

correctly, the following snapshot should appear. Take a look: see Figure 3.

As shown, the program jumps to 0x4017F2 and follows the normal execution (in this example, the normal execution is only a message). If the data is not correct, a "Data are not correct...." message appears. Afterwards, control is transferred to address 0x40180C. Now, let's take a look at the stack: see Figure 4.

Due to special initializations, it's easy to locate the variable areas. We focus on address 0x22FF2C; this is a return address and we can be 90% sure this return address would be used for RET instruction at address 0x40180C. We put another breakpoint in this address for it to continue execution until this point as shown: see Figure 5.

Great!!! ESP points to address 0x22FF2C. This is our target!

What should we do next? We must overwrite this address with value 0x4017F2, addressing directly the normal part of the program. This entry in the stack is in an offset of 6 above our work areas. The program does not check values in parameter so if we changed the first parameter to a value of 26, we can overwrite this entry. The second value must be 0x4017F2 in decimal: 4200434. This image clarifies the settings: see Figure 6. So, we must see the message first, which is telling us that the input is not correct. Afterwards, because we have changed the value of the return address, messages will tell us your data are correct as follows: We can take advantage of a vulnerability without injecting code and the exploit works even while the program is running in a system with Data Execution Prevention.

One Step More...

The explained technique above is only one way for exploiting a buffer overflow but there are other ways.

Another way is called *return-to-libc*. With ret2libc, we change the return address with the address of a system function and its parameters. Usually a calling to `system()` function. The latter technique I had explained is called return chaining. We see with an example.

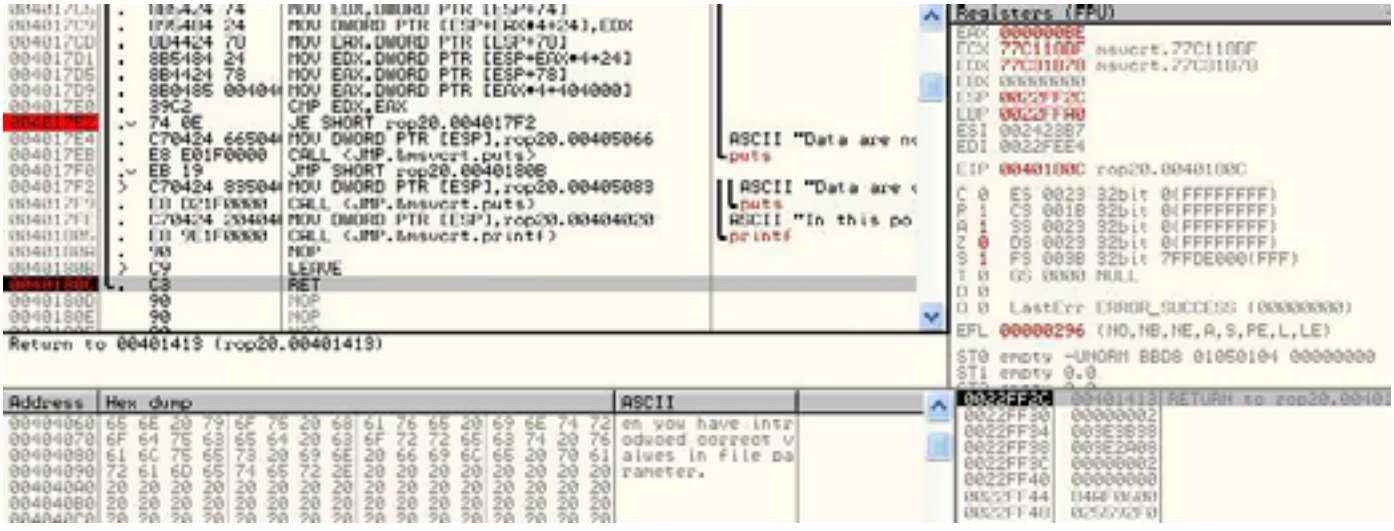


Figure 5.

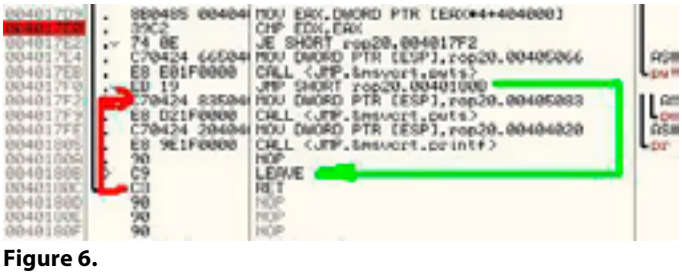


Figure 6.

Look at the following Figure 8.

We have identified the following instructions, each one is followed by RET instruction:

- pop a
- pop c
- mov [ecx], eax.

```
Data are not correct exiting
Data are correct entering
In this point you have entered in program flow accesible only when you have intr
oduced correct values in file parameter.
```

Figure 7.

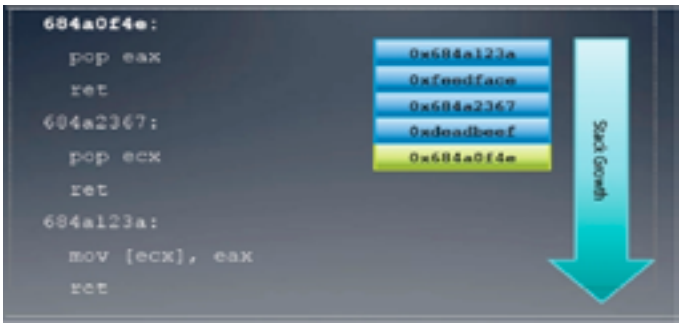


Figure 8.

Also, there is a RET leading program at the address: `0x684a0f4e`.

```
684a0f4e:
  pop eax
  ret
```

These instructions extract value on the top of the stack. And the following RET extracts value which transfers control to:

```
0xdeadbeef
```

Code at this address is:

```
684a2367:
  pop ecx
  ret
```

As anterior set of instructions, after extracting value from the stack and loading in the ECX register transfers control to this code:

```
684a123a:
  mov [ecx], eax
  ret
```

The final result will be as in the following figure:

References

1. "Options That Control Optimization", *https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html#Optimize-Options*
2. "Ollydbg is a powerfull debugger", *http://www.ollydbg.de*

This set of values is called "gadget"; a patient hacker can recollect a large set of instructions' addresses followed by a ret and make a catalogue. Then, by combining the needed values, he can execute instructions as if the code was being injected.

We can see gadgets like notes written by criminals in old movies:

Conclusion

In this article, I introduced how easy a hacker can exploit a stack overflow in an NX bit protected system and the other protections that we must not neglect as well such as compiler options and *Address Space Layer Randomization* (ASLR). Only when these protections are working together, we must think about a hardened programming environment.

JUANMA MENÉNDEZ

Juanma Menéndez is a system engineer with a strong experience in programming with a wide range of languages and operating systems. He is currently working as a Senior Consultant at Atos Spain. Juanma is also the author of z3r0 r0ws (http://zerorows.blogspot.com.es), a blog specialized in security and system programming. He can be reached via LinkedIn: http://es.linkedin.com/in/juanmamenendez/.

DEBUGGING AND TROUBLESHOOTING FUN, PROFIT AND GO HOME EARLIER

WORKSHOP ABSTRACT

DEBUGGING/TROUBLESHOOTING IS A REALLY USEFUL SKILL WHEN YOU ARE WORKING ON MAINTAINING LEGACY APPLICATIONS MAKING SOME SMALL INCREMENTAL CHANGES TO AN OLD CODE BASE, WHERE THE CODE HAS BEEN TOUCHED BY SO MANY HANDS OVER THE YEARS THAT IT'S REALLY BECOMING A MESS. SO, MANAGEMENT HAS DECIDED THAT THE CODE WORKS AS IT IS AND YOU ARE NOT ALLOWED TO CHANGE IT ALL OVER "THE RIGHT WAY (TM)".

IN THIS TUTORIAL, I'M GOING TO PRESENT A REAL LIFE SITUATION WHERE DEBUGGING SKILLS WILL SAVE US TIME, HEADACHES AND POSSIBLY FIND A SOLUTION DOING THE MINIMAL AMOUNT OF EFFORT.

WORKSHOP REQUIREMENTS

- BASIC KNOWLEDGE OF UNIX/LINUX SYSTEMS

WORKSHOP AGENDA

FIRST, WE ARE GOING TO DEBUG AN OLD *LEGACY C APPLICATION* THAT TAKES PLAIN TEXT FILES AND INSERTS THEM INTO A DATABASE UNTIL SOME NEW CHANGE MADE SOME DORMANT "FEATURE" AVAILABLE FOR THE USERS (DATA IS GETTING TRUNCATED AND USERS ARE COMPLAINING, THIS NEEDS TO BE FIXED ASAP BEFORE CLOSE OF BUSINESS). FOR THIS SITUATION WE ARE GOING TO USE THE CLASSIC "GDB" DEBUGGER.

SECOND, WE ARE GOING TO DEBUG A JAVA APPLICATION THAT IS HAVING PERFORMANCE ISSUES (TAKES 4 TIMES AS LONGAS THE OLD C APPLICATION). IT IS THE NEW WAY OF DOING THINGS INSTEAD OF THE OLD C APPLICATION THAT GETS THE DATA INTO THE DATABASE. UNFORTUNATELY, BELIEVE IT OR NOT, IT HAS THE SAME ISSUE AS THEIR OLD C COUNTERPART. WE WILL USE HEAP DUMPS, JDB AND GDB FOR THIS.

IN THE THIRD SCENARIO, WE WILL GO BACK TO THE FIRST ONE BUT WE WILL TAKE A DIFFERENT APPROACH. WE WILL DEBUG WITHOUT HAVING ACCESS TO THE SOURCE CODE AND ONLY RELAY IN THE DISASSEMBLED CODE WE COULD SEE THROUGH GDB.

FOR THE FOURTH SCENARIO, WE WILL ONLY HAVE A HEAP DUMP AND WE WILL NEED TO GO ALL THE WAY TO FIND THE ISSUE LURKING IN THE JAVA CODE.

FINALLY, WE WILL APPROACH BOTH SITUATIONS USING DTRACE WHICH IS AVAILABLE IN FREEBSD, OSX, SOLARIS AND OPENSOLARIS AND WILL CHECK IF THIS TOOL IS BENEFICIAL AND A TIME SAVER IN THE PROCESS..

INSTRUCTOR BIO

CARLOS NEIRA HAS WORKED SEVERAL YEARS AS A C/C++ DEVELOPER AND KERNEL PORTING AND DEBUGGING ENTERPRISE LEGACY APPLICATIONS. HE IS CURRENTLY EMPLOYED AS A C DEVELOPER UNDER Z/OS, DEBUGGING AND TROUBLESHOOTING LEGACY APPLICATIONS FOR A GLOBAL FINANCIAL COMPANY. ALSO HE IS ENGAGED IN INDEPENDENT RESEARCH ON AFFECTIVE COMPUTING . IN HIS FREE TIME HE CONTRIBUTES TO THE PC-BSD PROJECT AND ENJOYS METAL DETECTING.

IF YOU WANT TO PARTICIPATE IN OUR BSD ONLINE WORKSHOPS, PLEASE CONTACT UROS DRNOVSEK AT UROS.DRNOVSEK@BSDMAG.ORG DIRECTLY.

Among clouds Performance and Reliability is critical

Download syslog-ng Premium Edition product evaluation here

Attend to a free logging tech webinar here

BalaBit IT Security

www.balabit.com

syslog-ng log server

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

HIGH-SPEED RELIABLE LOGGING

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

UPDATE
NOWWITH
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

2012 Computing Security Awards
WINNER
Network Security Solution of the Year

www.titania.com