MAGAZINE

BSD

FOR NOVICE AND ADVANCED USERS

Overriding LIBC Functions

FreeBSD Desktop with Xfce, SLiM and i3lock

Automating Vulnerability Scanning with Vuls

Your Information Is Out There Ready for Buying

Processing Data in Parallel Using Multithreading

The TrueNAS Unified Storage X10

Interview with David Mytton

VOL 11 NO 07
ISSUE 07/2017 (95)
ISSN 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...

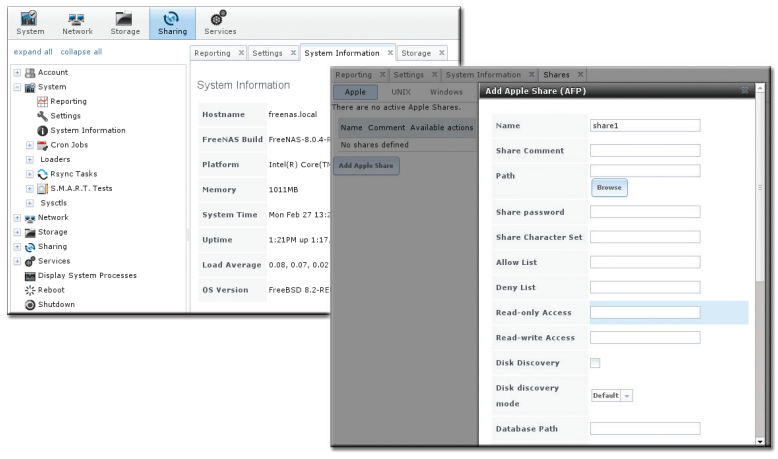
MAKE IT EASY ON YOURSELF

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/

Editor's Word

Dear Readers,

And we meet again. I hope that all of you are doing well, and enjoying your holidays. I also hope that you've missed us, and therefore you have allocated some time to read the new BSD issue. This time, we, the BSD reviewers' team, the authors and I want to present a few good readings. I am convinced you will find them very interesting.

This issue includes the article on *Overriding LIBC Functions To Do Constructive Things,* written by Rafael Santiago de Souza Netto. In his article, Rafael will introduce you to the main aspects of function override techniques in the C library from userspace. He will mainly focus on doing constructive things with function overrides, particularly in getting the most out of your system to solve problems. The next article that we included is written by Abdorrahman Homaei. He will tell you more about FreeBSD Desktop Environments and about Xfce, Xfce Pros and Cons. He goes further to describe what SLiM, i3lock are, and he teaches you how to Install Xfce, SLiM, and i3lock.

If you want more how-to articles, be ready to read the article written by Teppei Fukuda and Kota Kanbe entitled *Automating Vulnerability Scanning with Vuls*. They will introduce to you to Vuls (https://github.com/future-architect/vuls), an open-source vulnerability scanner for Linux/FreeBSD, which is agentless and written in Go. Vuls tells you which servers and software are related to the newly disclosed vulnerabilities. This leads us to the security topic. In my opnion, security is an important topic nowadays which is hard not to share some insights on it. In this issue of the BSD magazine, you will also get a chance to read the article, *Your Information Is Out There Ready to Be Bought,* authored by Jacob Alexander. He encourages you to read his article as a lesson about your security and of course, do your own research before hopping on the crazy train. At least you will know that all he wrote was based on facts, and your information is out there ready to be bought.

In this issue, you can continue reading about TrueNAS X10. Steve Wong, in this second part, will discuss four of the many use cases supported by the TrueNASX10. This is to help organizations work more effectively and efficiently, and to provide the rationale on why the X10 may be perfectly aligned with their IT environment.

For those who want to learn more about Unix, we have the article on *Processing Data in Parallel Using Multithreading* by Mark Sitkowski C.Eng, M.I.E.E Cybersecurity Consultant. Here are a few words from the abstract just to entice you to read the full text: *Real life being what it is, despite the isolation provided by the partitioned stack, the situation can still arise, where we might need to prevent more than one thread from executing a given function, or other block of code. For instance, we may be in the process of assembling a linked list, and it would be counter-productive to have two threads adding an element to the end of the list at the same time.*

At the end, please read the *Interview with David Mytton* to find out why and how he built his first product to have a simple way to get metrics and alerts from his servers.

So, let's read!!! And have a fun on our holidays!
Best regards,
Ewa & The BSD Team

TABLE OF CONTENTS

IN BRIEF

FreeBSD 11.1 Released

The FreeBSD Release Engineering Team is pleased to announce the availability of FreeBSD 11.1-RELEASE. This is the second release of the stable/11 branch.

Some of the highlights:

- Clang, LLVM, LLD, LLDB, and libc++ have been updated to version 4.0.0.
- Many third-party (contributed) software updates, such as the Elf Tool Chain, ACPICA, libarchive(3), ntpd(8), unbound(8), and more.
- Support for blacklistd(8) has been added to OpenSSH.
- The zfsbootcfg(8) utility has been added, providing one-time boot.config(5)-style options for zfsboot(8).
- The efivar(8) utility has been added, providing an interface to manage UEFI variables.
- Support for Microsoft® Hyper-V™ Generation 2 virtual machines has been added.
- The ena(4) driver has been added, providing support for "next generation" Enhanced Networking on the Amazon® EC2™ platform.
- The NFS client now supports the Amazon® Elastic File System™ (EFS).
- The EFI loader can now access remote files via TFTP in addition to NFS as a runtime configuration option.
- ZFS now stores compressed data in cache, improving cache hit rates and performance.
- Several updates to provide build reproducibility.

For more information about FreeBSD release, please visit:
https://www.freebsd.org/releases/11.1R/announce.html

#ServerEnvy: TrueNAS X10

The TrueNAS X10 is the 3rd generation of the TrueNAS unified storage line.

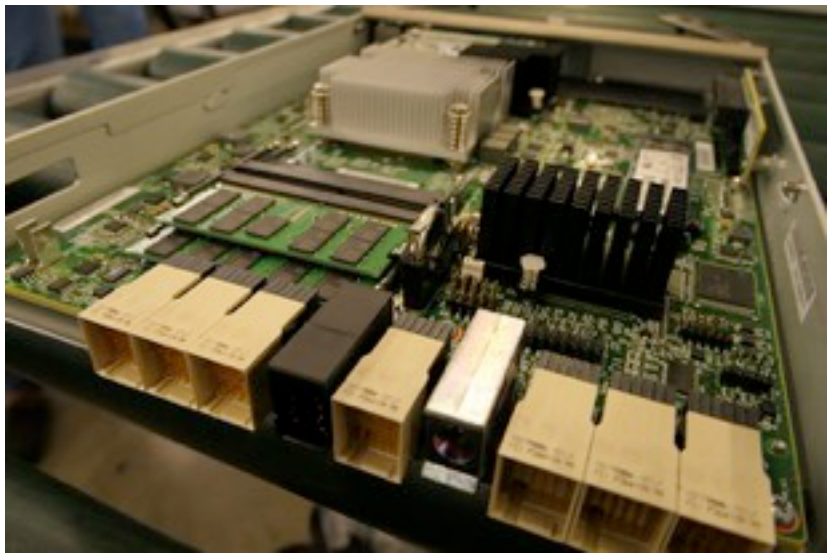
The X10 is the first of a new TrueNAS series, and will be expandable to up to 360TB with the TrueNAS ES12 expansion shelf.

This blog will show what makes the TrueNAS X10 unique.

Those looking for unified, highly available (HA), reliable enterprise grade storage with a self-healing file system will find that the TrueNAS X10 fits their business needs, at an economical price point.

Many businesses have ended up purchasing storage that far exceeded their requirements, or opted to purchase less reliable storage due to budget constraints. Many legacy storage vendors use 5-10 year old technology.

The TrueNAS X10 represents the birth of a new type of storage array and is ideal for businesses with RoBos (remote office / branch office) and enterprise workloads such as backup, replication, file sharing, and other applications.



The X10 is cost effective, retailing at a 30% lower price point than the Z20, making it an effective addition to your backup/DR infrastructure. The street price of a 20TB non-HA model falls under $10K. It's designed to move with six predefined configurations that match common use cases.

The dual controllers for high availability are an optional upgrade to ensure business continuity and avoid downtime.

The X10 boasts of a 36 hot swap SAS using two expansion shelves, for up to 360TB of storage, allowing you to backup thousands of VMs or share tens of thousands of files. One of the use cases for TrueNAS X10 is for backup, so users can upgrade the X10 to two ports of blazing 10GigE connectivity. The 20TB non-HA model enables you to backup over 7,000 VDI VMs for under $3.00 per VM.

Overall, the X10 is a greener solution than the TrueNAS Z product line, with the non-HA version boasting of only 138 watts of power and taking up only 2U of space.

Don't forget in-line compression, deduplication, unified file and block sharing, flash-assisted read & writing caching and its self-healing file system.

The X10 also features non-disruptive software upgrades, snapshots, clones, replication, thin provisioning, online capacity expansion, RAID protection, and the list goes on.

Best of all, the TrueNAS X10 starts at $5,500 street. You can purchase a 120TB configuration today for under $20K street.

The TrueNAS product line has customers ranging from the media and entertainment, government, to education industries.

You can learn more about the X10 through our official press release or read Steve's blog on the technical merits.

Source:
https://www.ixsystems.com/blog/serverenvy-truenas-x10/

DigitalOcean Releases Block Storage, Enabling Developers and Businesses to Build and Scale Larger Applications

Highly available and flexible SSD-based storage enables teams of developers to scale production workloads, add disk space, and build distributed applications without changing Droplet size

DigitalOcean, the provider of cloud computing platform designed for developers, today released Block Storage, the number one product request from customers. Block Storage is a highly available and scalable SSD-based offering that will enable developers to easily attach extra disk space to DigitalOcean Droplets (cloud servers). DigitalOcean's Block Storage costs $0.10/GB per month and is based on provisioned capacity only, therefore there is no need for complicated formulas to determine the overall cost for transactions or IOPs limit.

DigitalOcean was founded in 2011 with the mission to simplify the complexities of infrastructure by offering one simple and robust platform for developers to launch and scale their applications easily. The company has taken a "developer first" mentality and is now evolving its platform to further support the business needs of the developer. According to Netcraft, DigitalOcean has become the second largest and fastest growing cloud computing platform in the total number of public facing apps and websites. More than 700,000 registered customers have launched more than 18 million Droplets combined on DigitalOcean, nearly doubling from 10 million six months ago.

"We set out on a mission to build a simple and robust cloud computing platform so that engineering teams can spend less time configuring and automating their infrastructure and more time focused on software development," said Ben Uretsky, co-founder and CEO, DigitalOcean. "By adding a highly performant Block Storage offering to our platform, developers can easily deploy and manage their SaaS applications and businesses as they scale. This is one step closer to building the next generation platform."

The Easiest Way for Developers to Attach and Scale Storage to Support Their Businesses and Applications

Block Storage is a complementary product to DigitalOcean's Droplets, which provide compute power and local storage, by making them more expandable and flexible. By attaching Block Storage to DigitalOcean Droplets via the control panel or API, developers can achieve the following benefits:

- High Availability: Block Storage stores data on hardware that is separated from the Droplet, replicating it multiple times across different racks and reducing the chances of data loss in the event of hardware failure.

- Scalable and Flexible: Developers can easily scale and resize SSD-based Block Storage volumes from 1GB to 16TB and move it between Droplets. The ability to attach extra Block Storage to Droplets dramatically increases their flexibility, enabling developers to scale applications, databases, binary assets, shared and distributed filesystems and more.

- Reliable and Secure: All the data is encrypted at rest and transmitted to the Droplets over isolated networks.

Supporting Quotes

- GitLab developer Patricio Cano said: "Storage is a very real challenge for big teams or teams that work with lots of data. Storing assets will always take a lot of space which usually forces people to prioritize what they want to store or spend increasingly more money for each additional unit of storage. Using GitLab's GitHost as an example, without Block Storage, the amount of disk space an instance has is tied together with the plan the customer selects. With Block Storage, customers can now choose the right plan based on their team's needs (amount of RAM and processors) without having to worry about limited disk space, because they can now add up to 16TB of extra storage. For GitLab and our customers this is great news. We're excited to see how Block Storage in conjunction with Git LFS will enable users to easily store and version control their files."

- Mesosphere Co-founder and Chief Architect Benjamin Hindman said: "Developers across all platforms want simple, high-performance cloud storage like what DigitalOcean offers with its new Block Storage service. We're excited to work with DigitalOcean as part of the open source DC/OS storage interest group, in an effort make Block Storage available for DC/OS users, and to improve the overall storage experience for developers."

Additional Resources

- Read the blog on Block Storage: https://www.digitalocean.com/company/blog/block-storage-more-space-to-scale

- Read the tutorial on how to use Block Storage: https://www.digitalocean.com/community/tutorials/how-to-use-block-storage-on-digitalocean

Source: https://www.digitalocean.com/company/press/releases/digitalocean-releases-block-storage/

pfSense 2.3.4-p1 RELEASE Available!

pfSense software version 2.3.4-p1 is now available! This is a maintenance/errata patch available by running an update from an existing installation and it does not have a standalone installer to download. Version 2.3.4-p1 includes security fixes in pfSense and in underlying software, such as OpenVPN, and other bug fixes. pfSense software version 2.3.4-p1 includes updates for OpenVPN mentioned in an earlier blog post. Updating to 2.3.4-p1 will update OpenVPN automatically without any extra intervention. The firewall will reboot after applying the update.

pfSense software is Open Source

For those who wish to review the source code in full detail, the changes are all publicly available in three repositories on Github:

Main repository - the web GUI, back end configuration code, and build tools.

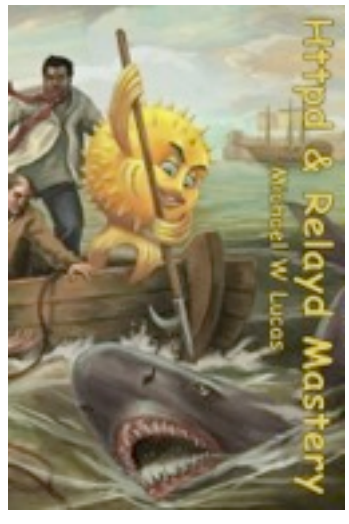
FreeBSD source - the source code, with patches of the FreeBSD 10.3 base.

FreeBSD ports - the FreeBSD ports used.

Source: https://www.netgate.com/blog/pfsense-2-3-4-p1-release-now-available.html

Relayd and Httpd Mastery by Michael W Lucas

“I think we’re gonna need a bigger web server.” OpenBSD has a solid reputation for security and stability. It’s well known for the OpenSMTPd mail server, the LibreSSL cryptography library, and the PF packet filter. But nobody ever talks about the load balancer, or the web server, until now.



The httpd web server provides a fast, stable, secure environment for your web applications. The relayd load balancer lets you distribute Internet application load across multiple hosts. Between the two, you can slash hundreds of thousands of dollars off the cost of building, deploying, and managing applications.
With Httpd and Relayd Mastery you’ll learn how to:

- set up websites
- configure software to run in a chroot
- run dozens or hundreds of sites on one host
- dynamically reconfigure sites with Lua patterns
- manage site logs
- maintain free, globally-valid SSL certificates
- improve performance with SSL stapling
- install and maintain two-server clusters
- distribute traffic between any number of hosts
- stop worrying about old SSL versions and bad crypto algorithms

Slash the amount of time you spend futzing with web servers. Get Httpd and Relayd Mastery today!

Source: https://www.createspace.com/7177283

TRUENAS

Introducing The TrueNAS Unified Storage X10. Part 2

Steve Wong, Director of Storage Product Management

Steve Wong is the director of product management at iXsystems. He is a senior level professional with over 20 years of experience in the fields of data communications, enterprise storage, networking, telecommunications, brand marketing, publishing, e-commerce and consumer package goods. Before iXsystems, Wong worked at SerialTek, Hitachi Data Systems, ClearSight Networks, Finisar, Anritsu and Mattel.

He began his career in investment banking at Bear Stearns, and then served as a member of technical staff at AT&T Bell Laboratories. Wong holds a BA from New York University and an MBA from the Kellogg Graduate School of Management, Northwestern University.

Four Use Cases for the TrueNAS X10

In this second part, I am going to discuss four of the many use cases supported by the TrueNASX10. This is to help organizations work more effectively and efficiently, and to provide the rationale on why the X10 may be perfectly aligned with their IT environment.

If you read part one of this blog on the TrueNAS X10 which was published a couple of weeks ago, you will recall that it focused mainly on the technical aspects of this new iXsystems platform. If you did not get a chance to read it, I highly recommend you do so now, and then return to this blog.

File Serving

In organizations, small or large, there is a requirement to deliver reliable and continuous data access to users and applications through a wide variety of protocols. The X10 is designed to be a high performance general purpose filer, providing users with simultaneous file data access critical for the successful performance of their business and work tasks. The X10 platform supports all the common file protocols including NFS, SMB and AFP. This means that if your organization has Linux or Unix desktops, the X10 has got you covered. If your users connect via a Windows computer, you are covered as well.

Even Apple systems can connect to data storage resources through AFP. iSCSI block access is also available for platforms such as Windows, VMware and various backup solutions.

Standard directory services such as Active Directory, NIS, Open Directory, and LDAP, support authentication and network resource services within multi-OS environments. Administrators can prevent users and specific user groups from consuming vast amounts of storage through the quota facility. Additionally, there is a myriad of data protection services available to safeguard data. For example, if a user inadvertently deletes some important files, these files can be restored within a matter of minutes.

The X10 runs TrueNAS and supports a High Availability option to ensure data access is continuously available. Users and applications can expect uptimes of 99.999%, meeting or exceeding even the most rigorous of service level agreements.

Disaster Recovery (DR)

For local data protection, many storage administrators rely on snapshots. The TrueNAS operating system is unique in that, snapshots happen nearly instantaneously and take up little to no space, so you can take many snapshots per hour without worrying about exhausting a snapshot pool. Also, TrueNAS does not impose any restrictions on how many snapshots can be taken.

My general advice to customers has been to take and keep as many snapshots as you can manage. Michael Dexter, a colleague of mine, recently wrote about how the snapshotting capability of TrueNAS can help you recover if you find yourself infected with a ransomware virus.

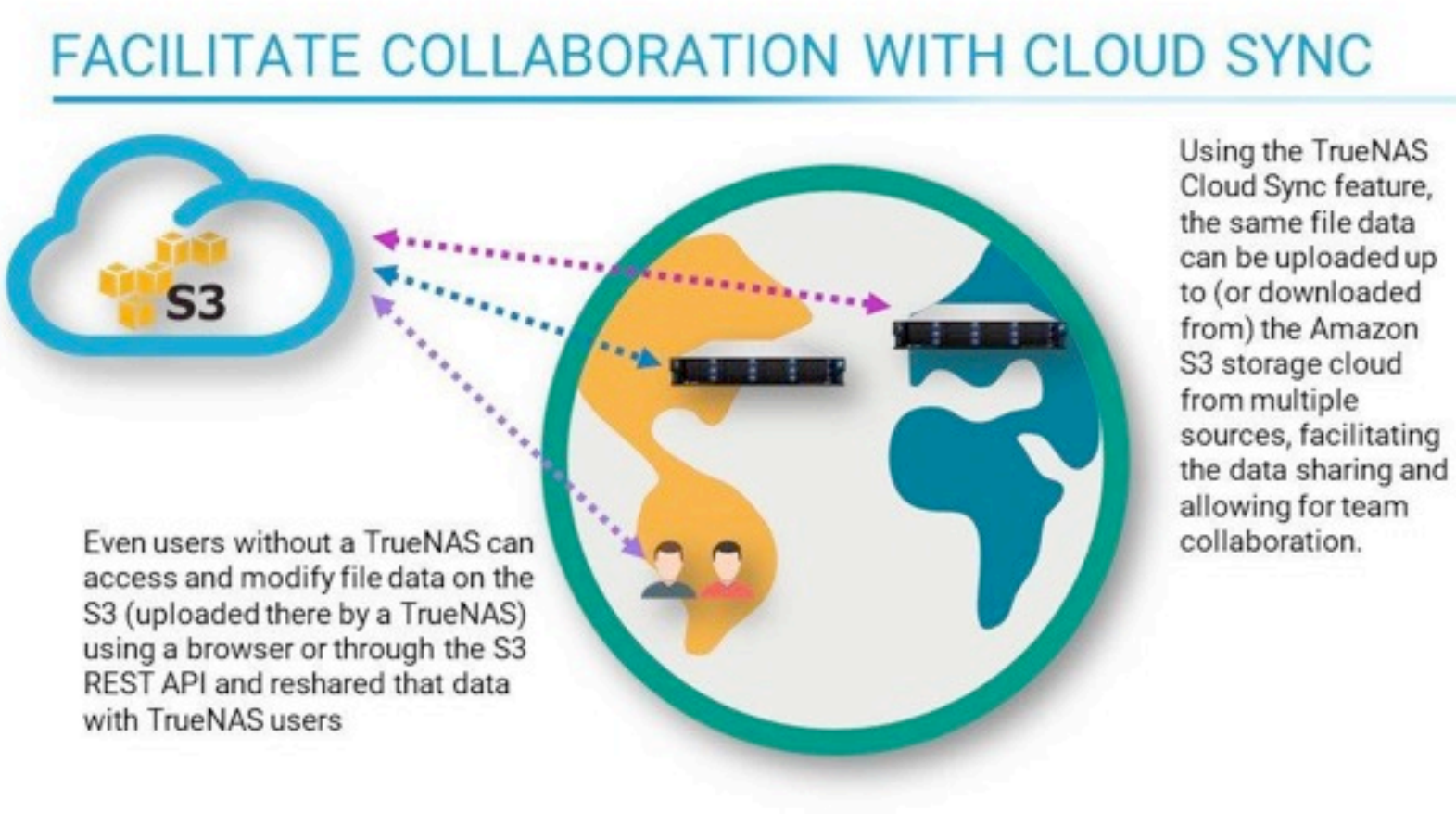
Remote replication is the process of copying or replicating data to another device, typically at a different location. A source system located in San Jose, California can replicate data to a secondary system located in Evanston, Illinois. Should some unforeseen disaster occur to the source system, the secondary system will have a complete backup of the data, allowing the business to quickly restore their business operations.

A TrueNAS X10 can replicate data to another TrueNAS system or any other OpenZFS-based system. TrueNAS replication uses snapshot-based technology. After the initial baseline replication from a source to a target, only “changes” are replicated afterwards. This greatly decreases the size of the backup window, and results in lower associated network cost because less bandwidth is consumed.

Cloud sync is a new capability that was introduced earlier this year by iXsystems. This is a TrueNAS built-in software feature that allows an administrator to use the Amazon S3 storage as a backup target. Specified data sets on a TrueNAS X10 can be replicated to an S3 cloud. Should something catastrophic occur, data can be restored from the S3 cloud back to a TrueNAS appliance. The Amazon S3 storage cloud is both resilient and persistent.

The cloud provider reports that the S3 is designed to provide 99.999999999% (11 9s if you are counting) durability of objects over any given year. This means that for every 10,000 objects that are stored, one object will be lost every 10,000 years – on average. In realistic terms, the odds of data loss occurring on the Amazon S3 are astronomically small.

There is even a use case for group collaboration and file sharing using the Cloud sync feature where file data can be shared amongst geographically dispersed teams and groups working on the same projects (see graphic below).



Backups

The TrueNAS X10 is designed to keep TBs of backup images. Its low cost per GB, in-line compression, and support of up to 360TB of raw capacity helps to improve ROI and makes the X10 a cost-effective platform for use as a replication and/or backup/archiving target.

Additionally, the OpenZFS file system was designed for data integrity and can self-heal itself, which keeps backup images safe. File system integrity is verified with checksums and if inconsistencies are found, those inconsistencies can be repaired automatically.

Edge to Core Applications

Many organizations have satellite or remote offices, sometimes called RoBos. These facilities often maintain and manage their IT infrastructure and run local services. The challenge for many of these organizations is how to equip their smaller environments with enterprise-ready storage solutions without having to compromise on a consumer-class solution.

A fully configured TrueNAS X10 with 20TB of enterprise storage starts at less than $10,000 (street pricing), striking the ideal balance between cost and capabilities. And as discussed in the DR section, data from a TrueNAS X10 appliance at a corporate satellite office can be replicated back to another TrueNAS at the corporate data center.

We discussed four use cases where the TrueNAS X10 excels. Businesses looking for a cost-efficient, fully enterprise-ready storage solution which supports a variety of use cases and aligns with their budgetary requirements should take a closer look at the TrueNAS X10.

Server U

Rack-mount networking server

Designed for BSD and Linux Systems

DESIGNED FOR FreeBSD | DESIGNED FOR GNU / Linux | DESIGNED FOR PRO apps FreeBSD Enterprise Appliance | DESIGNED FOR pfSense

Designed. Certified. Supported

Up to **5.5Gbit/s** routing power!

KEY FEATURES

- 6 NICs w/ Intel igb(4) driver w/ bypass
- Hand-picked server chipsets
- Netmap Ready (FreeBSD & pfSense)
- Up to 14 Gigabit expansion ports
- Up to 4x10GbE SFP+ expansion

PERFECT FOR

- BGP & OSPF routing
- Firewall & UTM Security Appliances
- Intrusion Detection & WAF
- CDN & Web Cache / Proxy
- E-mail Server & SMTP Filtering

contactus@serveru.us | www.serveru.us
8001 NW 64th St. Miami, LF 33166 | +1 (305) 421-9956

FreeBSD Desktop with Xfce, SLiM and i3lock

What you will learn ...

Introduction

Many people are looking for lightweight, fast and stable desktop environments.

However, a fully functional how-to is hard to come by.

In this article, a functional how-to consists of three elements which include:

- Desktop environments.
- Desktop manager (login manager).
- Desktop locker.

FreeBSD Desktop Environments

A desktop environment typically consists of icons, windows, toolbars, folders, wallpapers and desktop widgets.

A desktop environment aims to be an intuitive way for the user to interact with the computer using concepts which are similar to those used when interacting with the physical world, such as buttons and windows.

They are many, but the four commonly used desktop environments include:

GNOM

GNOME is a user-friendly desktop environment. It includes a panel for starting applications and displaying

status, a desktop, a set of tools and applications. Moreover, GNOME has a set of conventions that make it easy for applications to cooperate and be consistent with each other.

KDE

KDE is another easy-to-use desktop environment. This desktop provides a suite of applications with a consistent look and feel, a standardized menu and toolbars, keybindings, color-schemes, internationalization, and a centralized, dialog-driven desktop configuration.

Mate

MATE is a desktop environment forked from the now-unmaintained code base of GNOME 2.

Mate is lightweight, but it is not stable at all.

Xfce

Xfce is a fast, light, and efficient desktop environment based on the GTK+ toolkit used by GNOME. When compared to other environments, it is more lightweight and provides a simple, efficient, easy-to-use desktop. Moreover, it is fully configurable, has a main panel with menus, applets, and application launchers. It provides a file manager and sound manager, and is themeable.

More about Xfce

Xfce is a free and open-source desktop environment for Unix and Unix-like operating systems such as Linux, Solaris, and BSD.

Xfce aims to be a fast and lightweight desktop environment while maintaining its visually appealing and easy to use attributes. Xfce embodies the traditional UNIX philosophy of modularity and reusability. It consists of separately packaged parts that together provide all functions of the desktop environment.

Xfce Pros and Cons

Pros:

- Xfce is easy to install.
- Xfce is lightweight.
- Xfce is minimal but you can add what you want.
- Xfce is stable.
- Xfce code is clean.

Cons:

- There is no display manager.
- It doesn’t have a desktop locker.

What Is SLiM?

Simple Login Manager (SLiM) is a graphical display manager for the X Window System that can be run independently of any window manager or desktop environment. SLiM aims to be a light, completely configurable that is suitable for machines on which remote login functionalities are not required.

SliM can be used as Xfce display manager and it’s fully compatible with Xfce.

What Is i3lock?

i3lock is a simple screen locker like slock. After starting it, you will see a white screen (you can configure the color/an image). You can return to your screen by entering your password.

Features:

i3lock forks, so you can combine it with an alias to suspend RAM (run "i3lock && echo mem > /sys/power/state" to get a locked screen after waking up your computer from suspend to RAM)

You can specify either a background color or a PNG image which will be displayed while your screen is locked.

You can specify whether i3lock should bell upon a wrong password.

i3lock uses PAM, and therefore, it is compatible with LDAP, etc.

How To Install Xfce?

To install the Xfce package:

```
# pkg install xfce
```

Alternatively, to build the port:

```
# cd /usr/ports/x11-wm/xfce4
```

```
# make install clean
```

Unlike GNOME or KDE, Xfce does not provide its login manager to start SliM login manager, add its entry to ~/.xinitrc:

```
# echo "exec /usr/local/bin/startxfce4
--with-ck-launch" > ~/.xinitrc
```

How To Install SLiM?

In order to install SliM, issue the following commands:

```
# pkg install slim
```

Add these lines to /etc/rc.conf to load SliM at boot:

```
dbus_enable="YES"

hald_enable="YES"

slim_enable="YES"
```

Alternatively, you can issue the following commands:

```
# sysrc dbus_enable="YES"

# sysrc hald_enable="YES"

# sysrc slim_enable="YES"
```

How To Install i3lock?

i3lock installation is very easy:

```
# pkg install i3lock
```

For more information about i3lock, please issue this command:

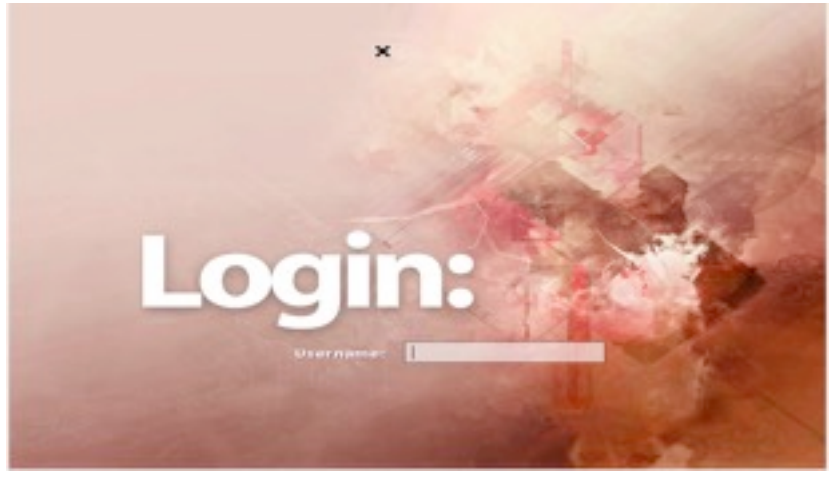
```
# man i3lock
```

Test All Together

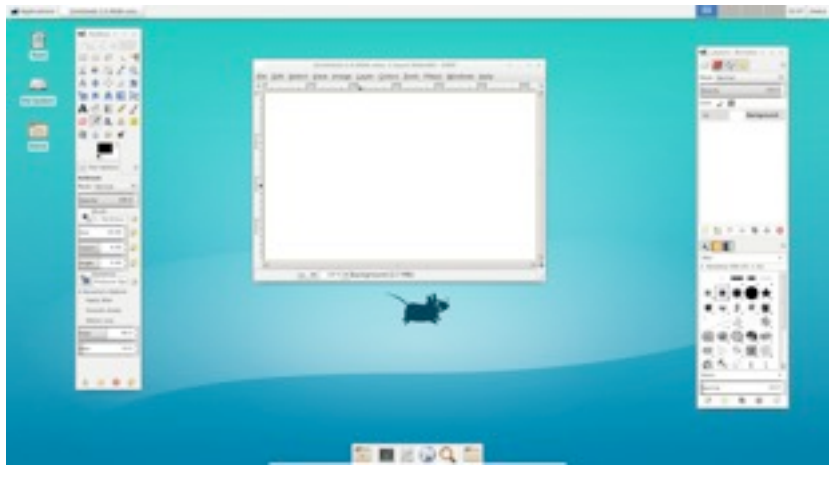
Reboot your machine to check if it works:

```
# reboot
```

After a while, you will see the SliM login manager:

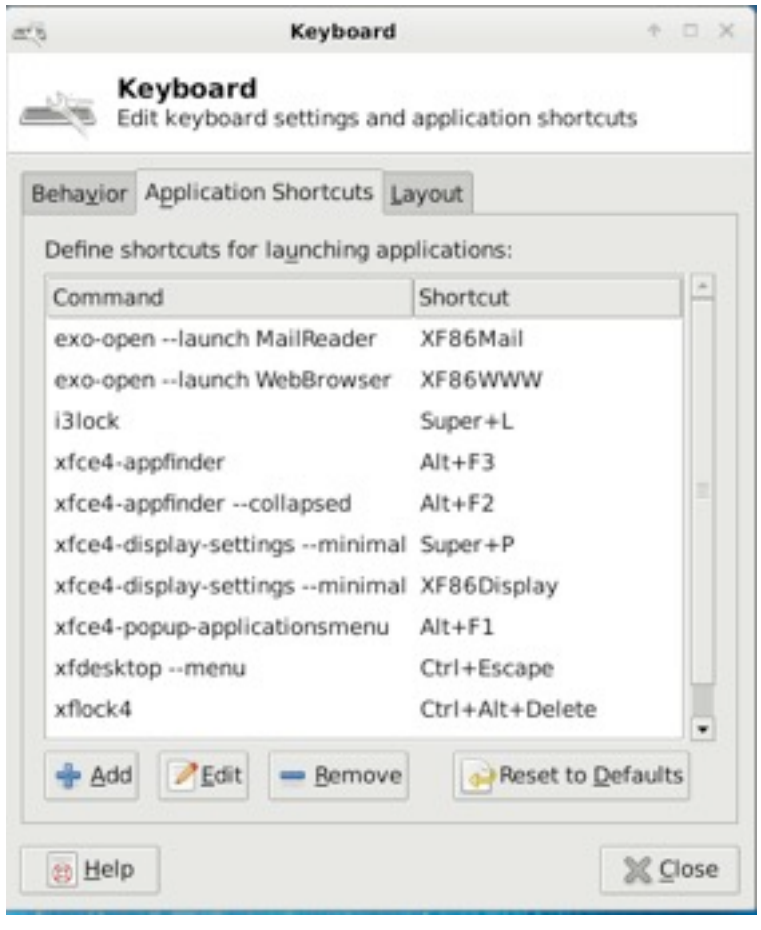


Now you can enter your Username and Password to login to Xfce desktop.



To test i3lock , you have to open a terminal from Xfce menu and issue the following commands:

```
# i3lock
```



Also, you can add i3lock to Xfce shortcuts.

As you can see, super+L(windows+L) will lock my desktop.

Conclusion

Xfce is a blazingly fast and stable environment platform, and there are much functionality you can add to your desktop.

Useful Links

http://meetbsd.ir

http://in4bsd.com

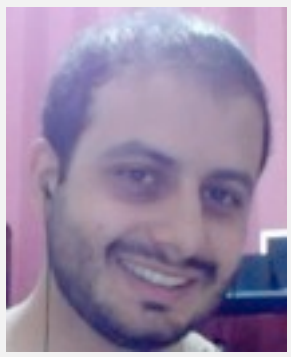
About the Author

Abdorrahman Homaei has been working as a software developer since 2000. He has used FreeBSD for more than ten years.

He became involved with the meetBSD dot ir and performed serious training on FreeBSD.

He is starting his own company in Feb 2017.

You can visit his site to view his CV: http://in4bsd.com



Overriding LIBC Functions To Do Constructive Things

This article will introduce the reader to the main aspects of function override techniques in the C library from userspace. Most of the existing material on this topic focuses on malicious uses of function overriding in malware. In this article, however, we will focus on doing constructive things with function overrides, particularly in getting the most of your system to solve problems. A basic knowledge of C will be necessary for following the text.

What is function overriding?

Function overriding is a technique that allows you replace some function code with another one. Usually, the new function will call the old one at some point in its execution. This is also known as “hooking”. As you can see, it can be a doorway to abyss, if you think about the countless nasty things that this technique could make possible. Indeed, many rootkits use this idea in some way. There are multiple ways of overriding a function. Here, we will focus on overriding a function during the linking phase in user mode.

You will see that even though function overrides are used by rootkits and other malware, the general technique can serve useful purposes and should not be demonized.

What are the main phases of a compiling process?

Some years ago, knowledge of the compilation process could be assumed. However, nowadays with tons of high-level interpreted languages, many new developers are not exposed to and are thus not familiar with compilation. Because of this, we will briefly discuss it here.

According to Aho, Sethi and Ullman, in their definitive text on compilers known as the “Dragon Book”, a compiler can be divided into six phases: lexical analyzer, syntax analyzer, semantic analyzer, intermediate analyzer, intermediate code generator, code optimizer and code generator. Output from each phase feeds the next one. All phases can access the symbol table generated from the

original source code and also can trigger errors. If an error occurs, it typically, aborts the whole process.

At a higher level, a typical language-processing system is made up of four main parts: preprocessor, compiler (covered in the previous paragraph), assembler, and loader/link-editor. Linking obviously occurs inside the last component.

As you can see, the language-processing system described in the Dragon Book is pretty close to the C compilation paradigm that C programmers are familiar with.

The preprocessor will handle the macros, the compiler will check and produce target assembly code, and the assembler will generate object files. The linker/loader will replace the relocatable codes by absolute machine code.

Relocatable codes are great

When you start a process on your system, all addresses referenced by this process have been previously determined. For instance, when your CPU needs to perform a jump operation, the address of this jump was previously calculated. Otherwise, it could cause a process termination due to invalid memory access. For example, if you jump to an address in the data section, the process will attempt to execute something that is not executable. Fortunately, the protected mode of your CPU disallows this sort of thing.

If we could only use absolute addresses in our programs, all software would need to be compiled into a huge monolithic software image. In practice, real programs use tons of libraries.

Libraries are a good example of relocatable code. The nice thing about this type of code is the possibility of re-using the same code in more than one software project.

Relocatable codes use relative address, so when the linker/loader handle them, these relative address are adjusted to absolute addresses and then the code can execute in the target machine.

Types of linking

There are two types of linking or two different “linkage models”. The difference between them is related to the time that relocatable codes are resolved to absolute addresses.

The dynamic link adjusts the relocatable addresses before executing the software. It involves use of the dynamic loader from the operating system, usually called “ld”.

The static link adjusts all relocatable addresses before generating the final executable file. It involves joining all codes (program and any libraries) into a single monolithic piece of software.

Software can mix the two types of linking. For example, a program can load at run-time a dynamic library. In this case, all relocatable codes are resolved on-the-fly during the dlopen() and dlsym() calls.

In UNIX-like environments, roughly speaking, when you use libraries with a “.a” extension, you are dealing with static linking. Whereas, when you use libraries with a “.so” extension, you are dealing with dynamic linking.

How can be possible override a function?

The technique used to accomplish this task depends on the linkage model used during the software building.

In UNIX-like environments, for dynamic libraries, there is an environment variable that allows you override functions. This variable is called “LD_PRELOAD”.

When you pass to the LD_PRELOAD some paths of shared libraries. Any library referenced in LD_PRELOAD is loaded before any other library, including the libc.

When the loader (ld) or even an explicit call to dlsym is done, the libraries searched at first will be those that are loaded. In this case, that would be the libraries referenced by LD_PRELOAD.

In the shared library where the function override is done, the way to find the original (overridden) function is by calling dlsym with a special handle. This handle is called RTLD_NEXT. Usually, it is the value “-1”. The RTLD_NEXT instructs the dlsym function to look for the next symbol in the chain. So the original function will be found.

Code Listing 1 shows a basic way of overriding the function fopen() from a shared library. The new fopen function will call printf with some information about the file, and then call the original fopen function.

The compilation of Code Listing 1 can be done in the following way:

```
gcc -shared override.c -oliboverride.so
```

The Code Listing 2 is the shell script that performs the function override using the LD_PRELOAD technique. Basically, you wrap your run command with:

```
LD_PRELOAD=<list of libraries> command
<args>
```

If you run a program using the shell script in Code Listing 2, any fopen() call will be overridden by the function in Code Listing 1. As a result, you will see the data related with path and mode printed in your terminal screen. For example:

```
./run.sh ./your_fopen_test file.txt
```

The output generated by the command above would be something like:

```
file_path = file.txt

mode = r
```

For static linking, the technique is almost the same for dynamic. However, since there is no LD_PRELOAD in the static link context, we would need to recompile the software with the library that contains all function override stuff.

Doing some useful stuff with these overrides

In this section, I will show you parts of a library that I use for writing C unit tests in my spare time projects. This library has an additional feature related with memory leak detection. The way that leaks are detected is by using a function overriding with a static library, in this case, the own unit test library. I call this library “cutest”.

Code Listing 1: Overriding the fopen function with a shared library.

```c
// File: override.c
#include <stdio.h>
#include <dlfcn.h>

#ifndef RTLD_NEXT
# define RTLD_NEXT -1
#endif

FILE *(*tru_fopen)(const char *file_path, const char *mode) = NULL;

void *handle = (void *)RTLD_NEXT;

FILE *fopen(const char *file_path, const char *mode) {
    if (tru_fopen == NULL) {
        tru_fopen = (void *)dlsym(handle, "fopen");
    }

    if (tru_fopen == NULL) {
        printf("ERROR: fopen hook not done!\n");
        return NULL;
    }

    printf("file_path = %s\nmode = %s\n", file_path, mode);

    return tru_fopen(file_path, mode);
}
```

Code Listing 2: Preloading the generated shared library.

```
#!/usr/local/bin/bash

APP=$1
shift
ARG=$1
LD_PRELOAD=./liboverride.so $APP $ARG
```

The Code Listing 3 shows the basic layout of a unit test written using libcutest. The macros are merely abstractions of boring function definitions, thus, you can focus only in what matters: testing your stuff. The code of your test goes inside the sections delimited by "CUTE_TEST_CASE(case_name)" "CUTE_TEST_CASE_END". Inside those sections, you can perform several types of assertions. The most basic is the "CUTE_ASSERT(<logic assertion>)", when false it will abort the test execution. More details are outside the scope of this article.

Once you build the unit test application, all that you should do is run this application and watch for the exit code. Non-zero exit codes indicate errors and a zero code indicates that all tests were passed.

A neat thing is that if you pass the option "--cutest-leak-check=yes", the application will automatically start monitoring the memory allocations. At the end if there are some allocated data still not freed, this will be reported and the application will exit with a non-zero code.

In other words, the library is capable of detecting memory leaks. A memory leak can be pretty harmful if your application stays running for hours, days or even years. This kind of resource consumption can slow down the system and also affect other processes.

Earlier detection (during unit test execution) could mitigate this kind of issue and also avoid disgusting remote debug sessions, crash dump analysis, etc. In the next sections, I will describe how I implemented the memory leak detection using simple function overrides.

Code Listing 3: The basic layout of a unit test application.

```
1  #include <cutest.h>
2
3  /*your includes goes here*/
4
5  CUTE_TEST_CASE(case_x)
6      /*your test stuff goes here*/
7      unsigned long x = 0;
8      CUTE_ASSERT(func_x(&x) < 0 || x == 0xabc);
9  CUTE_TEST_CASE_END
10
11 CUTE_TEST_CASE(case_y)
12     /*your test stuff goes here*/
13     CUTE_ASSERT(func_y() == 1);
14 CUTE_TEST_CASE_END
15
16 CUTE_TEST_CASE(tests_runner)
17     CUTE_RUN_TEST(case_x);
18     CUTE_RUN_TEST(case_y);
19 CUTE_TEST_CASE_END
20
21 CUTE_MAIN(tests_runner);
```

Code Listing 4: The function overrides done by libcutest.

```
1  void init_memory_func_ptr() {
2  #ifdef _WIN32
3  (...)
4  #endif
5      if (tru_calloc != NULL && tru_malloc != NULL &&
6          tru_realloc != NULL && tru_free != NULL) {
7          return;
8      }
9  #ifndef _WIN32
10     void *handle = (void *)RTLD_NEXT;
11     tru_calloc = (void *)dlsym(handle, "calloc");
12     tru_malloc = (void *)dlsym(handle, "malloc");
13     tru_free = (void *)dlsym(handle, "free");
14     tru_realloc = (void *)dlsym(handle, "realloc");
15     g_memhook_init_done = 1;
16 #else
17 (...)
18 #endif
19 }
```

Overriding the basic memory handling functions

Now that you already know the basics of function overriding will be easy to present to you the overrides done by libcutest. This action is performed inside the “cutest_memory.c” module. The overridden libc functions are: malloc(), calloc(), realloc() and free(). The Code Listing 4 shows the initialization function “init_memory_func_ptr()”; this function is responsible for overriding all relevant libc functions. In Code Listing 4, you will only see the code for UNIX-like systems, the Windows code was omitted. If you are interested in seeing how this works on Windows, you can clone the library repo and look at the source.

There is nothing new in Code Listing 4. The RTLD_NEXT technique was used and the flag “g_memhook_init_done” is a way of signaling that initialization function was called.

Now let’s see the code for the overridden malloc in Code Listing 5. In this code there are some nullity checks in order to avoid null pointer access. However, the basic idea is to call the original malloc function and to register allocation if the leak system is currently activated (“--cutest-leak-check=yes”). The pointer “g_cute_mmap” is simply a linked list that stores the initial address of the allocated memory and also the size of this allocation.

The code of the calloc is almost the same as the code for malloc, except, of course, the original function call. In the case for calloc is tru_calloc.

Code Listing 5: The malloc’s override.

```
1  void *malloc(size_t size) {
2      void *retval = NULL;
3      if (tru_malloc == NULL) {
4  #if defined(_WIN32) || defined(__FreeBSD__)
5          init_memory_func_ptr();
6  #endif
7          if (g_memhook_init_done) {
8              cute_log("libcutest INTERNAL ERROR: null tru_malloc().\n");
9          }
10         if (tru_malloc == NULL) {
11             return NULL;
12         }
13     }
14     retval = tru_malloc(size);
15     if (g_cute_leak_check) {
16         g_cute_mmap = add_allocation_to_cute_mmap_ctx(g_cute_mmap, size,
17                                                       retval);
18     }
19     return retval;
20 }
```

Code Listing 6: The realloc’s override.

```
1  void *realloc(void *ptr, size_t size) {
2      void *retval = NULL;
3      if (tru_realloc == NULL) {
4  #if defined(_WIN32) || defined(__FreeBSD__)
5          init_memory_func_ptr();
6  #endif
7          if (g_memhook_init_done) {
8              cute_log("libcutest INTERNAL ERROR: null tru_realloc().\n");
9          }
10         if (tru_realloc == NULL) {
11             return NULL;
12         }
13     }
14     retval = tru_realloc(ptr, size);
15     if (g_cute_leak_check) {
16         g_cute_mmap = rm_allocation_from_cute_mmap_ctx(g_cute_mmap, ptr);
17         g_cute_mmap = add_allocation_to_cute_mmap_ctx(g_cute_mmap, size,
18                                                       retval);
19     }
20     return retval;
21 }
```

The realloc's override before registering the allocation address take care of removing the possible previous allocation, since the realloc() function can change the original pointer address. The Code Listing 6 shows the realloc's override.

All pointers allocated by the functions malloc(), calloc() and realloc() should be released by the function free(). Thus, the free function's override should take care of removing any allocation previously registered by these three functions (if the leak system is activated). Take a look in the Code Listing 7.

How the leaks are detected at the end of the execution

The overridden allocation functions are responsible for registering their allocations into a global list and the free() function is responsible for removing the related released allocation from this global list. Since you release all you have allocated at the end of the execution this global list must be equals to NULL or in other words empty. If the pointer representing the list's head is non-null a report based in the remaining allocations contained into this list is shown and the program exits with an error code.

The code for detecting the presence of some memory leak is defined into the macro CUTE_MAIN(). In Code Listing 8, you can see the snippet of this part of the code.

Sometimes to find a memory leak can become hard...

I think that debugging should be the last resort. If your code is straightforward and clear, you can find a bug just by observing its behavior.

If for fixing up every single thing, you need a debugger it can be assumed as a code smell.

Anyway in cases like that, for memory leaks, the libcutest also includes a feature. This feature will launch a SIGTRAP when a specific leak id is hit.

A SIGTRAP is also known as a debug break. So if you are with the debugger attached in the process, it is possible to see the call stack and inspect the code more

Code Listing 7: The free's override.

```
1  void free(void *ptr) {
2      if (tru_free == NULL) {
3  #if defined(_WIN32) || defined(__FreeBSD__)
4          init_memory_func_ptr();
5  #endif
6          if (g_memhook_init_done) {
7              cute_log("libcutest INTERNAL ERROR: null tru_free().\n");
8          }
9          if (tru_free == NULL) {
10             return;
11         }
12     }
13     if (g_cute_leak_check) {
14         g_cute_mmap = rm_allocation_from_cute_mmap_ctx(g_cute_mmap, ptr);
15     }
16     if (ptr == NULL) {
17         return;
18     }
19     tru_free(ptr);
20 }
```

Code Listing 8: The memory leak detection logic.

```
1 (...)
2 if (g_cute_leak_check && g_cute_mmap != NULL) {
3     cute_log_memory_leak();
4     del_cute_mmap_ctx(g_cute_mmap);
5     exit_code = 1;
6 }
7 (...)
```

deeply. Of course, we need to recompile the application with debug information (the GCC's -g option gives you that).

Maybe you are questioning about how you will know the "leak id". Well, each memory leak report is numbered and you can see these data in the leak report. So you should use this value to cause a debug break exactly when that allocation occurs when you re-run the process.

To cause a debug break at a particular leak id, it is necessary to run the unit test program with the option "-–cutest-leak-id=<id>" besides the "--cutest-leak-check=yes".

Code Listing 9 shows how this debugging feature is implemented in libcutest. The parts related with Windows were omitted. According to the Code Listing 9, the function call "raise(SIGTRAP)" triggers a debug break if the current allocation id is equal to the allocation id informed by the user via command line option.

Some action with these overrides…

For now on, I will show you how a function override can be useful instead of destructive. Code Listing 10 introduces a tiny sample project called "greeting", this project is composed of three files. All that the resulting program does is display a greeting to the user based on the passed command line argument.

As you can see, the greeting function is kind of stupid because it allocates memory without needing it for doing the task. However, it was intentional.

For compiling those listed files, you should proceed as follows:

```
gcc greeting.c main.c -ogreeting
```

Running the program is pretty straightforward:

```
./greeting "john doe"
```

The Code Listing 11 includes a unit test project, using libcutest. This is based on the project presented in Code Listing 10.

Assuming that libcutest was pre-built inside a subdirectory called "cutest", the compilation command in supported BSDs would be:

```
gcc test_main.c -ogreeting-unit greeting.o
-Icutest/src -Lcutest/lib -lcutest
-lpthread -lexecinfo
```

Code Listing 9: How a debug break is raised in libcutest.

```
1  struct cute_mmap_ctx *
2      add_allocation_to_cute_mmap_ctx(struct cute_mmap_ctx *mmap,
3                                      size_t size, void *addr) {
4      struct cute_mmap_ctx *head = NULL;
5      struct cute_mmap_ctx *p = NULL;
6      if (g_cute_last_ref_file == NULL) {
7          return mmap;
8      }
9  #ifndef HAS_NO_PTHREAD
10     pthread_mutex_lock(&mmap_mutex);
11 #endif
12     head = mmap;
13     if (head == NULL) {
14         new_cute_mmap_ctx(head);
15         p = head;
16     } else {
17         p = get_cute_mmap_ctx_tail(mmap);
18         new_cute_mmap_ctx(p->next);
19         p = p->next;
20     }
21     p->id = ++g_cute_mmap_id;
22     p->size = size;
23     p->addr = addr;
24     if (p->id == g_cute_leak_id) {
25         raise(SIGTRAP);
26     }
27 #ifndef HAS_NO_PTHREAD
28     pthread_mutex_unlock(&mmap_mutex);
29 #endif
30     return head;
31 }
```

For executing the tests in the simplest way, you should run:

```
./greeting-unit
```

It will produce the following output:

```
-- running greeting_func_tests...

Hello you!!!

-- passed.

*** all tests passed. [1 test(s) ran]
```

Nice, all test passed. Now, let's check for memory leaks by re-running the tests with the "`--cutest-leak-check=yes`" option:

```
./greeting-unit --cutest-leak-check=yes
```

Now, the following output was produced:

```
-- running greeting_func_tests...

Hello you!!!

-- passed.

*** all tests passed. [1 test(s) ran]

cutest INTERNAL ERROR: Memory leak(s)
detected!!

>>>

Id=1 Address=0x085A0020
File=/root/src/override/test/unit/test_mai
n.c [The last check before leak was at
line #7] < Hello you!!!.. > 14 byte(s).

Leak total: 14 byte(s).

<<<
```

Code Listing 10: A tiny sample project compound by three files.

```
// File: greeting.h
#ifndef GREETING_H
#define GREETING_H 1

int greeting(const char *name);

#endif

// File: greeting.c
#include "greeting.h"
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

int greeting(const char *name) {
    char *str = NULL;

    if (name == NULL) {
        return 1;
    }

    str = (char *) malloc(11 + strlen(name));

    if (str == NULL) {
        return 1;
    }

    sprintf(str, "Hello %s!!!\n", name);

    printf("%s\n", str);

    return 0;
}

// File: main.c
#include <stdio.h>
#include "greeting.h"

int main(int argc, char **argv) {
    if (argc == 1) {
        printf("use: %s <data>\n", argv[0]);
        return 1;
    }
    return greeting(argv[1]);
}
```

As you can see, a memory leak was reported and just by looking at the data inside the memory it is possible to guess where this leak is occurring. However, let's assume that you could not figure out where this leak is happening.

In this case, we need to debug. The first thing to do is recompile the codes with debug information. The GCC "-g" option does the job. So:

```
gcc -g greeting.c main.c -ogreeting

gcc -g test_main.c -ogreeting-unit
greeting.o -Icutest/src -Lcutest/lib
-lcutest -lpthread -lexecinfo
```

Now, we need to re-run the tests with the debugger attached on it. The way of doing that with GDB is as follows:

```
gdb ./greeting-unit
```

After loading the binary data, the GDB will present you an output like the this:

```
GNU gdb (GDB) 7.11.1

Copyright (C) 2016 Free Software
Foundation, Inc.

License GPLv3+: GNU GPL version 3 or later
<http://gnu.org/licenses/gpl.html>

This is free software: you are free to
change and redistribute it.

There is NO WARRANTY, to the extent
permitted by law.  Type "show copying"

and "show warranty" for details.

Type "show configuration" for
configuration details.

For bug reporting instructions, please
see:

<http://www.gnu.org/software/gdb/bugs/>.

Find the GDB manual and other
documentation resources online at:

<http://www.gnu.org/software/gdb/documenta
tion/>.

For help, type "help".

Type "apropos word" to search for commands
related to "word"...

Reading symbols from
./greeting-unit...done.

(gdb)
```

The most important information is "Reading symbols from ./greeting-unit...done." because it indicates that the debug information were loaded, thus your debug session will be more sane.

To run the binary from inside the debugger is simple, all you should do is execute the command "run". However, the binary needs the "--cutest-leak-check=yes" option, in this case you should execute "run –-cutest-leak-check=yes".

After running it into GDB, the same leak report will be presented to you. Now we will re-run the unit test passing the leak id "1" in the following way:

```
run --cutest-leak-check=yes --cutest-leak-
id=1
```

This time the GDB output will be:

Code Listing 11: The unit test for the tiny sample project.

```
1  // File: test_main.c
2  #include <cutest.h>
3  #include "../greeting.h"
4
5  CUTE_TEST_CASE(greeting_func_tests)
6      CUTE_ASSERT(greeting(NULL) != 0);
7      CUTE_ASSERT(greeting("you") == 0);
8  CUTE_TEST_CASE_END
9
10 CUTE_TEST_CASE(test_main)
11     CUTE_RUN_TEST(greeting_func_tests);
12 CUTE_TEST_CASE_END
13
14 CUTE_MAIN(test_main);
```

```
Starting program:
/root/src/override/test/unit/greeting-unit
--cutest-leak-check=yes --cutest-leak-id=1

[Thread debugging using libthread_db
enabled]

Using host libthread_db library
"/lib/libthread_db.so.1".

-- running greeting_func_tests...

Program received signal SIGTRAP,
Trace/breakpoint trap.

0xb7f91e87 in raise () from
/lib/libpthread.so.0

(gdb)
```

The message “Program received signal SIGTRAP, Trace/breakpoint trap.” indicates that a debug break has occurred, so the process is interrupted. We can then inspect the callstack using the command “bt”. This command will produce the following output:

```
(gdb) bt
#0  0xb7f91e87 in raise () from
/lib/libpthread.so.0

#1  0x0804a20c in
add_allocation_to_cute_mmap_ctx ()

#2  0x0804a52d in malloc ()

#3  0x0804a677 in greeting (name=0x804a810
"you") at
/root/src/override/test/greeting.c:14

#4  0x08048e0c in greeting_func_tests ()
at
/root/src/override/test/unit/test_main.c:7

#5  0x08048ebc in test_main () at
/root/src/override/test/unit/test_main.c:1
1

#6  0x08049189 in main (argc=3,
argv=0xbfffef04) at
/root/src/override/test/unit/test_main.c:1
4

(gdb)
```

Inspecting the presented callstack, we see that the last thing done in our code occurred in the function “greeting()”, at the line 14 in the file “greeting.c”. After that, things occur inside malloc() and it comes from libcutest, do you remember of “add_allocation_to_cute_mmap_ctx()” function? Look it just after the top of the stack, but it is not relevant for us.

Let’s take a look in “greeting.c” in the Code Listing 12. The GDB is telling us that the last thing done inside our code before the debug break was at line 14.

Well, according to Code Listing 12, line 14 is about a malloc() call. If you inspect the code, the “str” pointer is neither being returned nor freed.

Code Listing 12: The greeting function.

```
1  // File: greeting.c
2  #include "greeting.h"
3  #include <stdlib.h>
4  #include <string.h>
5  #include <stdio.h>
6
7  int greeting(const char *name) {
8      char *str = NULL;
9
10     if (name == NULL) {
11         return 1;
12     }
13
14     str = (char *) malloc(11 + strlen(name));
15
16     if (str == NULL) {
17         return 1;
18     }
19
20     sprintf(str, "Hello %s!!!\n", name);
21
22     printf("%s\n", str);
23
24     return 0;
25 }
```

You have found the problem. Supposing you added a free(str) at end of the function. Thereafter, re-compile both the main project and the unit test project. When you re-run the unit-test with “--cutest-leak-check=yes”, no memory leak reports will be shown. This indicates that your code is freeing everything that it allocates.

Conclusion

This text was intended show you the useful and constructive side of function overriding. When it was done, it presented a way of detecting memory leaks using my unit test library.

Two main ways of performing function overrides from userspace were discussed. The first one is by using the LD_PRELOAD environment variable. This technique is useful when you cannot re-compile the program. The second one technique is the static way, and involves recompiling the software. The nice thing about a re-compile is portability, when do you need to use this technique in non-UNIX environments.

However, in UNIX-like systems, the LD_PRELOAD technique can also be used for detecting memory leaks in programs that you cannot re-compile. But in this case I personally prefer to use Valgrind (https://www.valgrind.org). When running my unit tests, usually, I like to use my memory leak system besides Valgrind. In this case, I can quarantine the problem as soon as possible and solve it while still in the building phase.

Much more can be done with memory profiling and function overrides. As instance invalid accesses to memory can be detected if you override functions like “memcpy”, “memmove”, etc. Also string functions like “strncpy”, “strcpy”, “strcat” and so on.

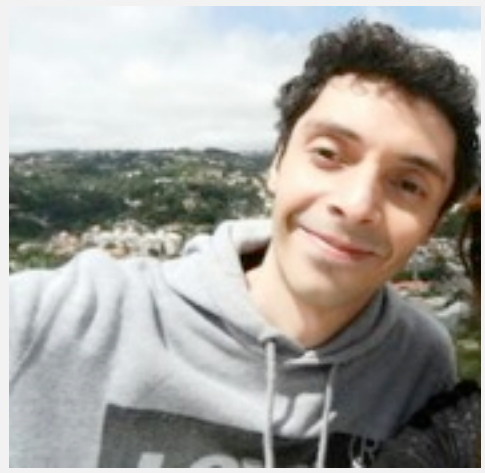
With LD_PRELOAD, it is also possible, for example, to add network proxy support to applications that do not offer this feature natively. The main idea for doing this task is override the main socket functions and of course, handle the proxy complications on your own.

If you are interested in knowing more about the unit test library used as sample in this text, you can download the source code at https://github.com/rafael-santiago/cutest. There you can also find instructions about how to build and use it.

As you can see, function overrides are great and you can use them for several useful and non-destructive tasks.

About the Author

Rafael Santiago de Souza Netto is a Computer Scientist from Brazil. His main areas of interest are Programming, Computer Networks, Operating Systems, UNIX culture, Compilers, Cryptography, Information Security, Social Coding. He has been working as Software Developer since 2000. You can find him at GitHub (as rafael-santiago).



BORN TO DISRUPT

MODERN. UNIFIED. ENTERPRISE-READY.

INTRODUCING THE TRUENAS® X10, THE MOST COST-EFFECTIVE ENTERPRISE STORAGE ARRAY ON THE MARKET.

Perfectly suited for core-edge configurations and enterprise workloads such as backups, replication, and file sharing.

- **Modern:** Not based on 5-10 year old technology (yes that means you legacy storage vendors)
- **Unified:** Simultaneous SAN/NAS protocols that support multiple block and file workloads
- **Dense:** Up to 120 TB in 2U and 360 TB in 6U
- **Safe:** High Availability option ensures business continuity and avoids downtime
- **Reliable:** Uses OpenZFS to keep data safe
- **Trusted:** Based on FreeNAS, the world's #1 Open Source SDS
- **Enterprise:** 20TB of enterprise-class storage including unlimited instant snapshots and advanced storage optimization for under $10,000

The new TrueNAS X10 marks the birth of a new entry class of enterprise storage. Get the full details at iXsystems.com/TrueNAS.

iXsystems™

Automating Vulnerability Scanning with Vuls

In recent years, the number of cyber security incidents has been steadily increasing. The importance of security measures is widely recognized both by the general public and among experts. However, most of the successful attacks originate from only a few sources. According to the Verizon Data Breach Investigations Report (DBIR) 2016, “The distribution is very similar to last year, with the top 10 vulnerabilities accounting for 85% of successful exploit traffic.” Despite the belief that most attacks are sudden and unpreventable, the attackers often use known vulnerabilities. So the application of existing security updates can easily prevent most successful attacks.

However, it is not easy for system administrators to apply security updates in a timely manner. To avoid downtime in a production environment, it is common for a system administrator to choose not to use the automatic update option provided by the package manager, and instead to perform updates manually. This leads to the following problems:

The system administrator will have to constantly watch out for any new vulnerabilities in the NVD (National Vulnerability Database) or similar databases.

It might be impossible for the system administrator to monitor all the software if there are a large number of software packages installed on the server.

It is time-consuming and onerous to determine the servers affected by new vulnerabilities. The possibility of overlooking an affected server is very real.

As a result, security updates are often delayed, leading to security incidents.

Consider a case where you actually deal with security updates. First, you check CVE (Common Vulnerabilities and Exposures, https://cve.mitre.org/) and NVD (https://nvd.nist.gov/) to collect vulnerability information. As an example, if you are dealing with CVE-2016-5636 (which is one vulnerability of Python), you need to check the specific pages on the above websites for CVE-2016-5636:

https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-5636

https://nvd.nist.gov/vuln/detail/CVE-2016-5636

Next, log in to the server you wish to examine:

```
$ ssh user01@freebsd01
```

Then, in order to know whether or not there is an impact from this vulnerability, collect the information of the installed software:

```
$ pkg info python27
python27-2.7.11
Name : python27
Version : 2.7.11
Installed on : Fri Mar 25 02:33:39 2016
UTC
Origin : lang/python27
Architecture : FreeBSD:10:amd64
Prefix : /usr/local
```

```
Categories : python lang ipv6
Licenses : PSFL
Maintainer : python@FreeBSD.org
WWW : http://www.python.org/
...
```

Since this vulnerability affects Python before v2.7.12, we can conclude that this server is affected by this vulnerability.

As seen above, these manual operations require a great deal of labor. If you manage hundreds of servers, it is almost impossible to check all servers. In this article, we introduce open-source software that solves these problems.

What's Vuls?

We will introduce you to Vuls, open-source vulnerability scanner for Linux/FreeBSD, agentless, and written in Go (https://github.com/future-architect/vuls). Vuls tells you which servers and software are related to the newly disclosed vulnerabilities. Vuls uses multiple detection methods including OVAL, Security Advisories, NVD, and changelog, and it is possible to scan many servers at high speed by using the parallel processing in Go language. It has the following characteristics.

- Informs users of the vulnerabilities that are related to the system.
- Informs users of the servers that are affected.
- Detects vulnerabilities automatically to prevent any oversight.
- Reports on a regular basis using CRON or other methods to manage vulnerability.

Vuls Features

- Scan for any vulnerabilities on Linux/FreeBSD Server.
    - Scan middleware that are not included in OS package management
- Scan middleware, programming language libraries and framework for vulnerability.
- Agentless architecture
    - User is required to only setup one machine that is connected to other target servers via SSH
- Nondestructive testing
    - Pre-authorization is not necessary before scanning on AWS
- Email and Slack notification is possible (supports Japanese language)

For details, refer to README (https://github.com/future-architect/vuls/blob/master/README.md)

Vuls Installation

There are two ways to scan vulnerabilities in Vuls.

A. Scan via SSH Mode (Remote Scan Mode)
B. Scan without SSH (Local Scan Mode)

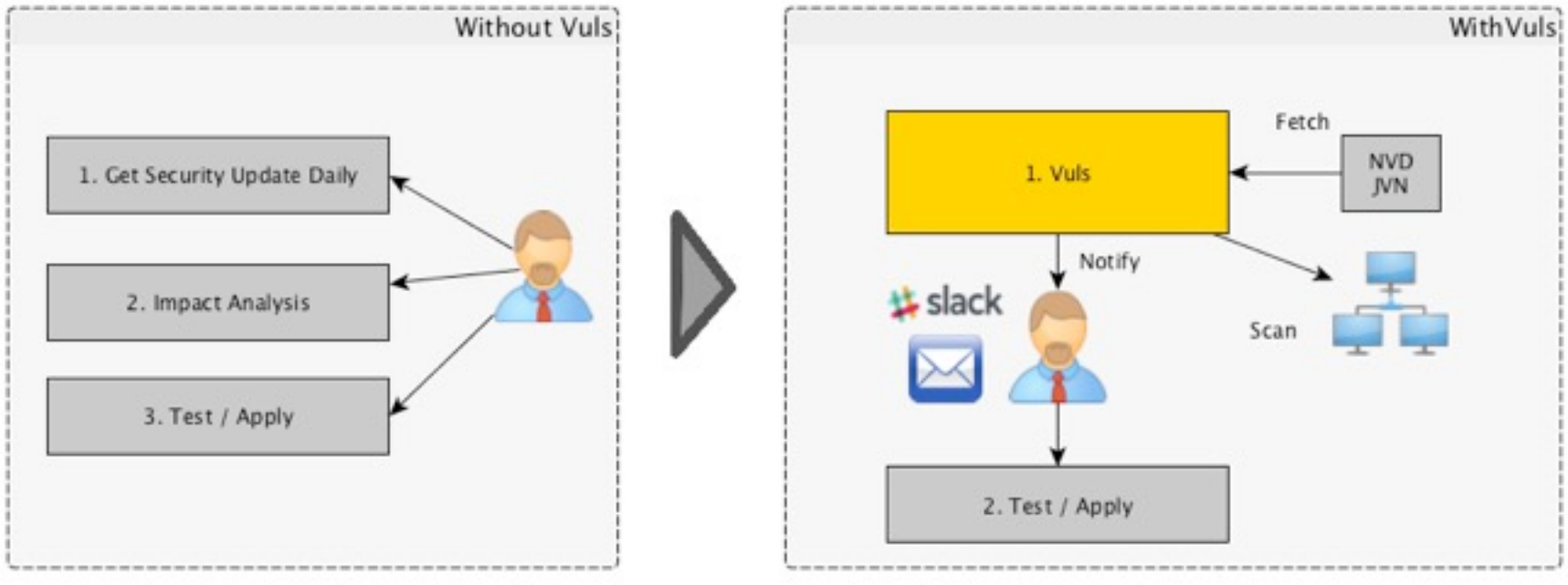


Figure 1. Vuls Motivation

Figure 2. Vuls Abstract

In this article, we will introduce mode A. Install Vuls on one server and connect to other servers from that server via SSH for scan.

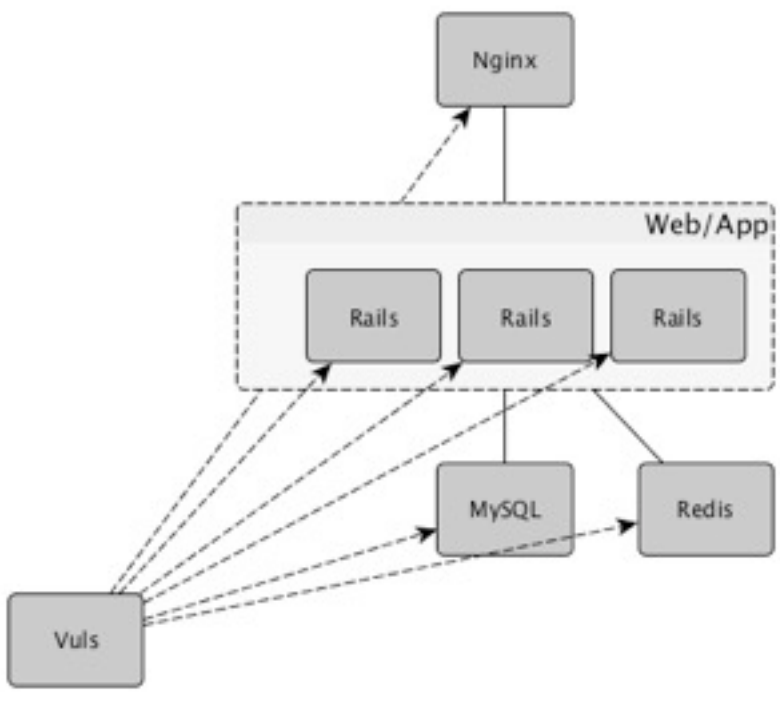


Figure 3. Vuls Use Case

As an example, we will setup one server for Vuls and connect to localhost via SSH. We use FreeBSD 10.3.

Create Account

Create a user account to run vuls.

```
root@:~ # adduser
Username: vuls
Full name: Vuls monitor user
Uid (Leave empty for default):
Login group [vuls]:
Login group is vuls. Invite vuls into
other groups? []:
Login class [default]:
Shell (sh csh tcsh git-shell nologin)
[sh]: csh
Home directory [/home/vuls]:
Home directory permissions (Leave empty
for default):
Use password-based authentication? [yes]:
Use an empty password? (yes/no) [no]:
Use a random password? (yes/no) [no]:
Enter password:
Enter password again:
Lock out the account after creation? [no]:
Username   : vuls
Password   : *****
Full Name  : Vuls monitor user
Uid        : 1002
Class      :
Groups     : vuls
Home       : /home/vuls
Home Mode  :
Shell      : /bin/sh
Locked     : no
OK? (yes/no): yes
adduser: INFO: Successfully added (vuls)
to the user database.
```

```
Add another user? (yes/no): no
Goodbye!
```

Install requirements

Install the required packages.

```
root@:~ # pkg install go git sudo sqlite3
gmake
root@:~ # su - vuls
$ mkdir $HOME/go
```

Add these lines into $HOME/.profile.

```
$ cat .profile
...snip...
export GOPATH=$HOME/go
export PATH=$PATH:$GOROOT/bin:$GOPATH/bin
```

Set the OS environment variable to current shell.

```
$ . .profile
```

Configure SSH

Vuls doesn't support SSH password authentication. So you have to use SSH key-based authentication. Create a keypair on the localhost, then append public key to authorized_keys on the remote host to be scanned (localhost, in this case).

```
$ ssh-keygen -t rsa -b 4096
```

Copy ~/.ssh/id_rsa.pub to the clipboard. Paste from the clipboard to ~/.ssh/.authorized_keys.

```
chmod 700 ~/.ssh
cat ~/.ssh/id_rsa.pub >> authorized_keys
chmod 644 authorized_keys
```

Vuls requires registering the scanner fingerprint in known_hosts before scanning, so you have to log in in advance. In this case, connect to localhost via ssh.

```
$ ssh vuls@127.0.0.1
The authenticity of host '127.0.0.1
(127.0.0.1)' can't be established.
ECDSA key fingerprint is
SHA256:L7NR7Uhq+Mnj9j43qIGzrKitMVtI5EnRLBb
RFLZFQUc.
No matching host key fingerprint found in
DNS.
Are you sure you want to continue
connecting (yes/no)? yes
Warning: Permanently added '127.0.0.1'
(ECDSA) to the list of known hosts.
FreeBSD 10.3-RELEASE-p18 (GENERIC) #0: Tue
Apr 11 10:31:00 UTC 2017

Welcome to FreeBSD!
```

Deploy go-cve-dictionary

go-cve-dictionary (https://github.com/kotakanbe/go-cve-dictionary) is open-source software used by Vuls.

go-cve-dictionary fetches vulnerability information from NVD and inserts them into SQLite3 or MySQL.

Make directory for Vuls logs.

```
root@:~ # mkdir /var/log/vuls
root@:~ # chown vuls /var/log/vuls
root@:~ # chmod 700 /var/log/vuls
```

Install go-cve-dictonary.

```
$ mkdir -p
$GOPATH/src/github.com/kotakanbe
$ cd $GOPATH/src/github.com/kotakanbe
$ git clone
https://github.com/kotakanbe/go-cve-dictio
nary.git
$ cd go-cve-dictionary
$ gmake install
```

The binary was built under $GOPATH/bin

Fetch vulnerability data from NVD. It takes about 10 minutes (on AWS).

```
$ cd $HOME
$ for i in `seq 2002 $(date +"%Y")`; do
go-cve-dictionary fetchnvd -years $i; done
...
$ ls -alh cve.sqlite3
```

Deploy Vuls

Install Vuls.

```
$ mkdir -p
$GOPATH/src/github.com/future-architect
$ cd
$GOPATH/src/github.com/future-architect
```

```
$ git clone
https://github.com/future-architect/vuls.g
it
$ cd vuls
$ gmake install
```

The binary was built under $GOPATH/bin

Configure Vuls

Create a config file(TOML format).

```
$ cd $HOME
$ cat config.toml
[servers]

[servers.freebsd01]
host         = "127.0.0.1"
port        = "22"
user        = "vuls"
keyPath     = "/home/vuls/.ssh/id_rsa"
```

Check Configuration

Check config.toml and settings on the server before scanning

```
$ vuls configtest
[Jul 12 00:35:08]  INFO [localhost]
Validating config...
[Jul 12 00:35:08]  INFO [localhost]
Detecting Server/Container OS...
[Jul 12 00:35:08]  INFO [localhost]
Detecting OS of servers...
[Jul 12 00:35:09]  INFO [localhost] (1/1)
Detected: freebsd01: FreeBSD
10.3-RELEASE-p19
[Jul 12 00:35:09]  INFO [localhost]
Detecting OS of containers...
[Jul 12 00:35:09]  INFO [localhost]
Checking dependencies...
[Jul 12 00:35:09]  INFO [localhost]
Checking sudo settings...
[Jul 12 00:35:09]  INFO [freebsd01] sudo
... No need
[Jul 12 00:35:09]  INFO [localhost]
Scannable servers are below...
freebsd01
```

Scan

Start scanning.

```
$ vuls scan
[Jul 12 00:35:36]  INFO [localhost] Start
scanning
[Jul 12 00:35:36]  INFO [localhost]
config: /home/vuls/config.toml
[Jul 12 00:35:36]  INFO [localhost]
Validating config...
[Jul 12 00:35:36]  INFO [localhost]
Detecting Server/Container OS...
[Jul 12 00:35:36]  INFO [localhost]
Detecting OS of servers...
[Jul 12 00:35:37]  INFO [localhost] (1/1)
Detected: freebsd01: FreeBSD
10.3-RELEASE-p19
[Jul 12 00:35:37]  INFO [localhost]
Detecting OS of containers...
[Jul 12 00:35:37]  INFO [localhost]
Detecting Platforms...
[Jul 12 00:35:49]  INFO [localhost] (1/1)
freebsd01 is running on other
[Jul 12 00:35:49]  INFO [localhost]
Scanning vulnerabilities...
[Jul 12 00:35:49]  INFO [localhost]
Scanning vulnerable OS packages...

One Line Summary
================
freebsd01       FreeBSD10.3-RELEASE-p19 22
CVEs 30 updatable packages
```

Report

Use the report subcommand to check the scan results. This result can be notified in several ways such as mail and Slack.

View one-line summary:

```
$ vuls report -format-one-line-text
-cvedb-path=$PWD/cve.sqlite3
[Jul 12 00:37:10]  INFO [localhost]
Validating config...
[Jul 12 00:37:10]  INFO [localhost]
cve-dictionary: /home/vuls/cve.sqlite3
[Jul 12 00:37:10]  INFO [localhost]
Loaded:
/home/vuls/results/2017-07-12T00:35:49Z

One Line Summary
```

```
================
freebsd01       Total: 22 (High:7
Medium:15 Low:0 ?:0)  30 updatable
packages
```

View short summary:

```
$ vuls report -format-short-text
-cvedb-path=$PWD/cve.sqlite3 --lang=ja
[Jul 12 00:38:05]  INFO [localhost]
Validating config...
[Jul 12 00:38:05]  INFO [localhost]
cve-dictionary: /home/vuls/cve.sqlite2
[Jul 12 00:38:05]  INFO [localhost]
Loaded:
/home/vuls/results/2017-07-12T00:35:49Z

freebsd01 (FreeBSD10.3-RELEASE-p19)
===================================
Total: 22 (High:7 Medium:15 Low:0 ?:0)  30
updatable packages

CVE-2016-5636   10.0 (High)     Integer
overflow in the get_data function in
zipimport.c in CPython (aka Python)
                                before
2.7.12, 3.x before 3.4.5, and 3.5.x before
3.5.2 allows remote attackers
                                to have
unspecified impact via a negative data
size value, which triggers a
                                heap-based
buffer overflow.

http://www.cvedetails.com/cve/CVE-2016-563
6

https://vuxml.freebsd.org/freebsd/1d0f6852
-33d8-11e6-a671-60a44ce6887b.html

python27-2.7.11_1 -> python27-2.7.13_6

Confidence: 100 / PkgAuditMatch
...
```

TUI

Vuls has Terminal-Based User Interface to display the scan result (see Figure 4).

```
$ vuls tui
```

If you got the following error, try another terminal such as xterm.

```
$ vuls tui

[Jul 13 13:39:36] INFO [localhost]
Validating config...
ERRO[0000] termbox: error while reading
terminfo data: termbox: unsupported
terminal
```

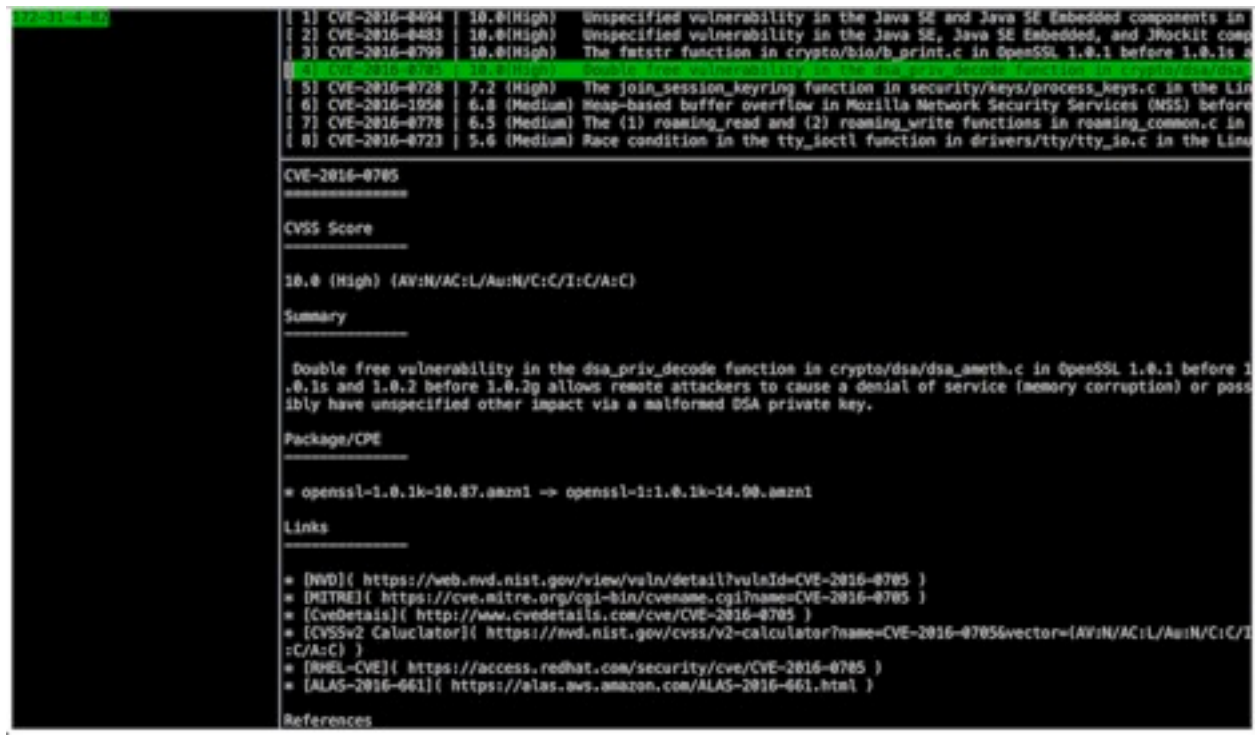


Figure 4. Terminal-Based User Interface

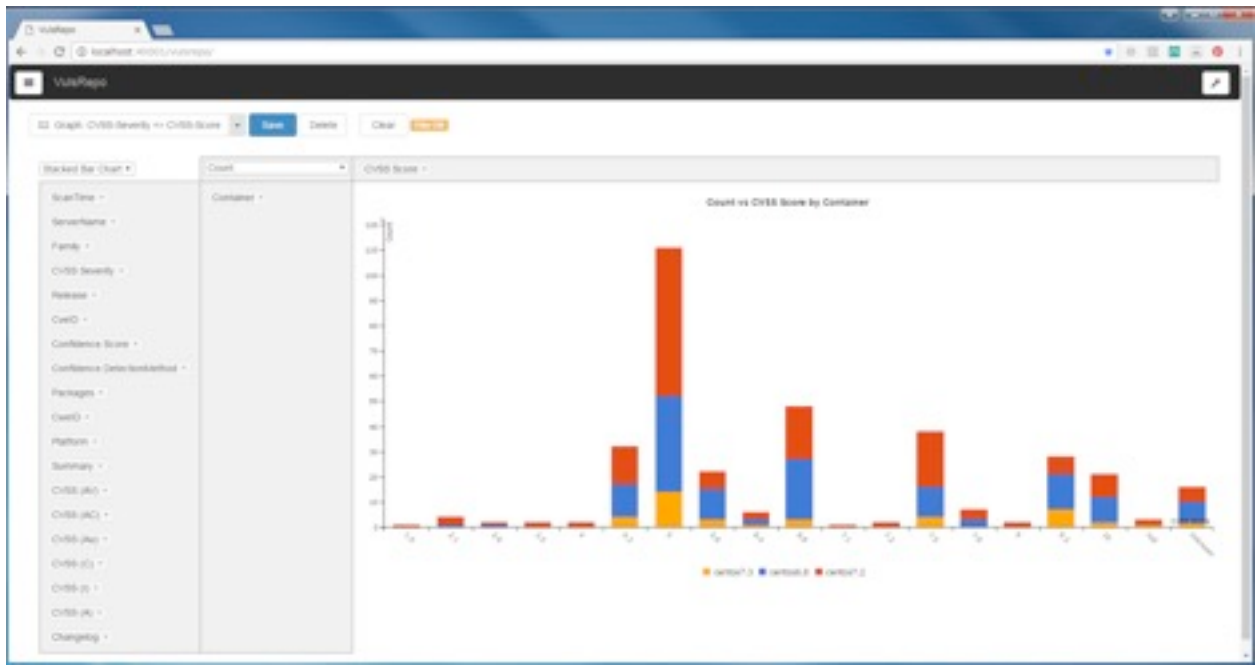


Figure 5. Count CVSS Scores

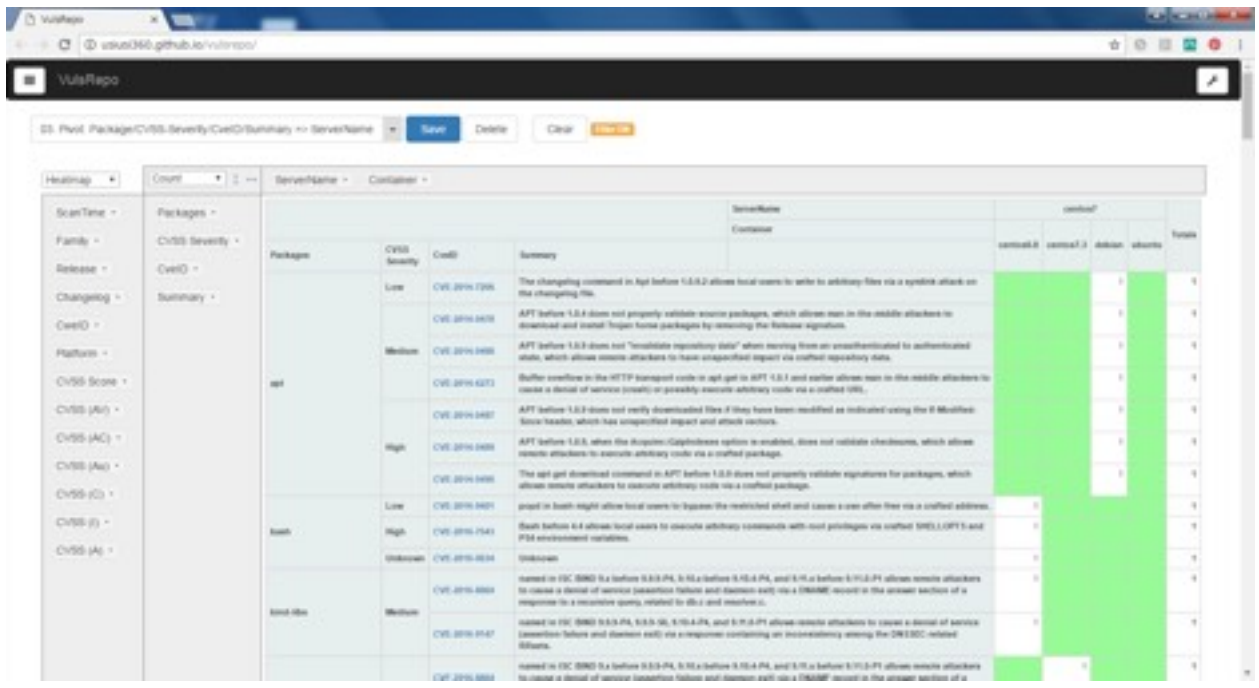


Figure 6. Make a list of CVE by Pivot Table

Web UI

VulsRepo (https://github.com/usiusi360/vulsrepo) is a Web UI for Vuls (see Figures 5 and 6).

Conclusions

Using Vuls makes it possible for the system administrators to automatically detect vulnerable servers. Since the system administrators can only focus on vulnerabilities related to the system they manage, the burden is greatly reduced. As a result, the security updates can be applied continuously. We, developers want Vuls to make the world peaceful.

About the Authors

Teppei Fukuda is the developer of Vuls and a security engineer at Future Architect, inc in Japan.
https://github.com/knqyf263

Kota Kanbe is the author of Vuls and a Senior Architect at Future Architect, Inc in Japan.
https://github.com/kotakanbe

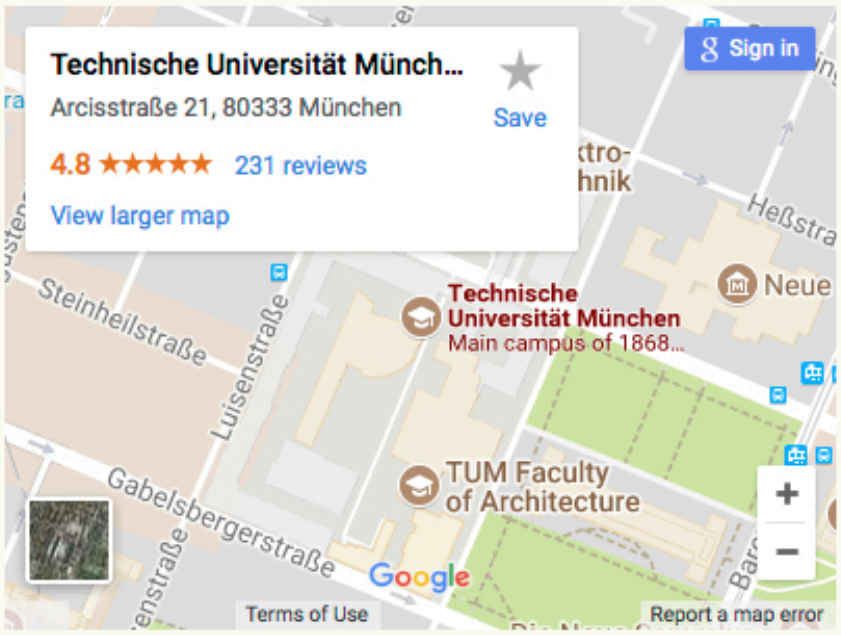
MINIXCon 2017 Announcement

After the successful MINIXCon 2016 in Amsterdam, the MINIXCon steering committee has decided to hold MINIXCon 2017 in Munich, Germany, on Sept. 18, 2017. The Chair of Operating System (Prof. Dr. Uwe Baumgarten) will be our patron from the Technical University Munich. The idea is to exchange ideas and experiences among MINIX 3 developers and users as well as discussing possible paths forward. Future developments will now be done like in any other volunteer-based open-source project. Increasing community involvement is a key issue here. We also will make a special effort to get industry involved.

The videos of the MINIXCon 2016 talks can be found here .

Date, Time, and Location

Monday, 18 September 2017 from 09:00 to 18:00 at the Theresianum, Technical University of Munich.



Giving a Talk

The call for presentations is now open. To propose a talk, please see the Call for Proposals page:

www.minix3.org/conference/2017/cfp.html

Registration

To attend the conference, you need to register here. The early registration fee is 50 euro (20 euro for students) for registrations before 4 Sept. 2017. After that, it is 75 euro for regular and 30 euro for students. This includes morning and afternoon coffee breaks, and lunch. The lunch will include a vegetarian option.

Transportation

You can fly to Munich Airport almost anywhere in the world. However, if you are coming from somewhere in Europe within 400 km of Munich, the train is probably faster and more convenient.

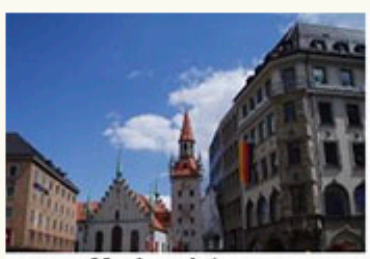
Hotels

Munich has over 400 hotels in a wide variety of price classes. Tripadvisor.com has ratings of almost 400 of them.

Dont be afraid about the prices of the hotels. The Bavarian Beerfast starts on Saturday before the MinixCon2017. The hotel that is physically closest to the campus is Hotel Königswache. You can get more affordable hotels reachable by subway (Stop Theresienstrasse, NO, not Theresienwiese!) If you plan to do some sightseeing in Munich before or after the conference, the Pinakotheken are reachable on foot and of course the Oktoberfest Another attraction are the Eisbach Surfers.

Munich Tourism

While MINIXCon 2017 is undoubtedly the biggest attraction in town, Munich has hundreds of things to do and places to see. Tripadvisor has a long list of Here are a few of the better-known ones.


Marienplatz


BMW museum


Town hall


Deutsches museum


Surfing the Eisbach wave

Your Information Is Out There, Ready to Be Bought

In today's age, our presence on the Internet is just as real and important as our offline presence. From social media to email to online banking, our lives are intertwined with the Internet in some form of fashion. Those that use the Internet daily for work, school or pleasure know the dangers of viruses and pop-ups, but what about information? How did we become so okay with giving our information out to websites? Did they do something to earn our trust, that they deserve to know our home address, our phone numbers, pictures of our children, and everything that we do at every hour of the day? Even if you pay all your bills in cash, never sign up for an online account or never even open a web browser, websites still know you very well because they can share information with brick and mortar businesses. Most people don't tend to think about these things and just give websites or businesses everything that they ask for. MBA@UNC explored whether people actually wanted to keep their data private and how much it meant to them. The results were from an Italian university's research in which participants would sell their smartphone activity for as little as two euros. (https://onlinemba.unc.edu/blog/data-brokers-infographic/) Now I know I am going to start sounding paranoid and I will probably ask you to put on a tin foil hat if you continue reading. But bear in mind that this is not just speculation, it is backed up by facts. I encourage you to check my work and do your research before hopping on the crazy train. If you don't learn anything from this, at least know that all of this is supported by facts, and your information is out there ready for the best bidder. There is a popular saying concerning the internet, and it goes, "if you're not the customer, you're the product". There are a few variations and a few people claim to be the original source. That aside, the meaning of this saying is that: companies need to make money to survive and they can make it in numerous ways. The obvious way is through directs sales, but with the increased use of the Internet, it's also possible for companies to make money through different means, particularly through the use of customers' data.

Google made 90 Billion dollars last year and they sell very few things. (https://www.google.com/finance?fstype=ii&q=NASDAQ:GOOG) Many services like Google make less money from physical items and more on either selling customer data or collecting, analyzing and using that data to make themselves money. Google, in particular, uses every search, email, location, picture and website you visit to help them build a complete profile of who you are and what things you like. They use this information to sell advertisement space to companies in places that you would most likely click on their advertisements. For instance, if you recently visited a travel agency website looking to book a trip to Jamaica, you may start seeing ads for similar trips on different websites. This is because Google is not limited to collecting data or presenting data on their site, any site that uses a Google product (ie. Google Analytics, Google Adwords, etc.) can be used to collect data on their behalf. (http://www.investopedia.com/stock-analysis/2012/what-does-google-actually-make-money-from-goog1121.aspx)

There are several other services that do this, but besides Facebook and a few others, not on the massive scale or as well as Google does it. Most of the information Google collects is only used internally, but what about the companies that actively sell your information? (http://www.newsweek.com/secretive-world-selling-data-about-you-464789) Well, there are way too many to list, but they all fall into some basic categories. The first being rewards programs from department stores, gas stations, grocery stores, etc. Yes, stores make money from making consumers believe that their rewards program will give them the cheapest deals. In reality, the prices are often marked up to make the discount seem bigger than it is, but that's not the only reason they have these programs. They aren't giving you discounts because they are being nice, no, they use the information you give them to make more money. (http://dfw.cbslocal.com/2015/08/26/rewards-cards-how-much-personal-information-is-collected/) Why else would they need to know your full name, date of birth, address, email address and phone number? They make massive amounts of money off this information because the promise of free or discounted items makes any sane person sign up immediately. The collected information is also not treated as important as financial information and is often outsourced to be handled by a third party. (https://www.consumeraffairs.com/news04/2005/loyalty_cards.html)

Third parties are a major weak link in the data collection process as they do not always have to follow the same terms of service that the read and agreed to. If the first party company uses an unreliable or untrustworthy third party, your personal information could be exposed. Not only that, the more moving pieces and places your data is stored, the more probable it will be that your data is exposed or stolen. Businesses get hacked daily and if everyone in the data collection process doesn't use strong security, the weak link will be the first to be attacked. SOHA did a study showing that 63% of data breaches result from attacks on third party vendors. (http://go.soha.io/hubfs/Survey_Reports/Soha_Systems_Third_Party_Advisory_Group_2016_IT_Survey_Report.pdf) With enough resources, any business can be hacked, but if good security is used all throughout the process, we can make it much less likely.

Data brokers have, perhaps, the most amount of information on individuals. There have been several stories covering this in the news, so I won't cover it too much in depth. Data brokers are the ones who sort most of the information that comes from online surveys, give-aways, credit companies, public records, warranty registrations, bankruptcy reports, etc. (http://www.npr.org/sections/alltechconsidered/2016/07/11/485571291/firms-are-buying-sharing-your-online-info-what-can-you-do-about-it) They are not limited to where they get their information and it can often get creepy, such as TLO and their license plate readers. (http://www.tlo.com/vehicle-sightings) These data brokers use this information to put you into a category and sell your information in very specific lists based on criteria such as age, gender, ethnicity, income level or medical history. (https://www.consumer.ftc.gov/blog/ftc-report-examines-data-brokers)

Websites know way more than you think just by collecting metadata such as user agent string, IP address, browser extensions, etc. In a process known as browser fingerprinting, your browser can be uniquely identified to be your browser. Even if you are accessing the site from a different location, if you are using the same browser, they still know it's you. This coupled with your home IP address, websites can determine who in your household visits their site, how often, and if they are vulnerable to online advertisements. How is this accomplished? The programs installed on your computer have a lot more access than they should and this is not limited to browsers. This includes access to your location, operating system version, CPU information, battery information, screen size and color depth, touch screen support, network addresses including your local IP and its subnet, gyroscope, installed fonts and more. Coupled with your browser version, settings, installed plugins and extensions, language, etc., a very accurate fingerprint can be made of your browser that is unique from every other web browser in the world. Even if you are using a VPN, a capable website can easily identify your online activities and distinguish you from other users. (http://webkay.robinlinus.com/ , https://panopticlick.eff.org/)

Social media is a completely different topic and can be almost entirely avoided by not using it, as long as no one in your family uses social media. With social media platforms such as Twitter, Facebook, Instagram, Snapchat, we freely give them all our information with very few questions asked. We do this to show our status in life, boost our self-confidence and to connect with friends and family, which we could do in person or over the phone just as easy. However, we're not going to focus on the human element of social media. We will focus on the large amount of data that social media can collect about you. With the use of third party cookies, extensions and social media scripts such as Facebook's like button,

social media sites can track your web browsing history while you are logged into their services and even when you're not logged in. (http://www.huffingtonpost.com/nate-hanson/how-to-stop-facebook-from_b_8160400.html)

Facebook, by itself, knows too much about its users to be comfortable. Facebook can determine sexual preference, your exact GPS location (as accurate as the room in your building), number of credit lines, brand of clothing your household buys, medications, when you are likely to move, etc. The list goes on and on with its endless amount of creepiness. (https://www.washingtonpost.com/news/the-intersect/wp/2016/08/19/98-personal-data-points-that-facebook-uses-to-target-ads-to-you/?tid=sm_tw) Facebook also operates one of the most accurate facial recognition software in the world, which unsurprisingly is more accurate than what the United States FBI uses. Facebook correctly identifies photos 98% of the time while the FBI is only accurate 85% of the time. This is, of course, due to their very large sample size of 1.28 billion active users. The FBI is much more limited in their sample size, so it makes sense that they do not get the same success rate. (http://www.npr.org/sections/alltechconsidered/2016/05/18/477819617/facebooks-facial-recognition-software-is-different-from-the-fbis-heres-why) This is just for Facebook; other social media platforms can figure out similar things about you as well. It should be noted that even if you have never had a social media account, your family members can violate your privacy pretty easily with their accounts. For instance, if Dave's spouse posts about a vacation they are going on with Dave and the kids, it's safe to assume Facebook will know that someone named Dave is going on vacation with the kids.

Sometimes even your own government will sell your information. In the United States, many Division of Motor Vehicles (DMV)'s make millions from selling your personal information. This includes your name, address, birth date, emergency contacts, etc. (http://www.wkbw.com/news/nys-dmv-made-60-million-selling-drivers-personal-information) I witnessed this first-hand when I moved and had to get a new driver's license. I provided a lot of personal information to the very nice and sincere woman who helped me get my driver's license, not expecting all that information to become public. A few weeks later the same personal information I gave to the USDMV was posted on Whitepages premium for all to see. Sometimes the local government doesn't want money in return, they want more data in exchange for the data they have in the case of Waze, Moovit and Strava in cities like Rio. (https://www.forbes.com/sites/parmyolson/2014/07/07/why-google-waze-helps-local-governments-track-its-users/)

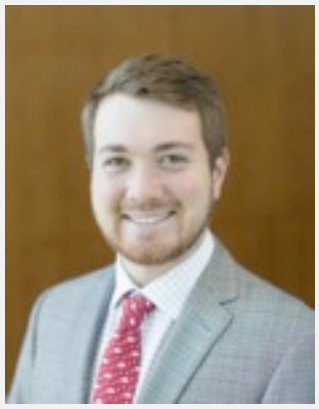
The big question is: why? Why do companies care what I do throughout my day? Why do they collect this massive amount of data on everyone? Like a lot of things, the answer comes down to money. Big data is such a profitable venture that many companies specialize in the field and make tons of money from it. There are two key issues I find with big data collection: the first being I like to live a private life and I do not enjoy my information being publicly posted on the Internet free for hackers, scammers, and telemarketers to take advantage of; the second being the collection and storage process of my personal information through a third party who doesn't care about the security of my personal information. You may not care where your personal information is being stored or how it's being used, but consider this: there is only one of you that lives at your address, that has your birthday and that likes the same things you do. If you become a target by an individual, a group, a government or whoever, you will be compromised. That could come in the form of character defamation, slander, death threats, job loss, identity fraud or identity theft. Some things are harder to overcome than others and it's better to be safe than sorry. In the case of identity theft, it could cost you a lot of time and money to repair the destruction to your life.

Now if you're like me, you are paranoid all of the time and you don't give any information to anyone that doesn't explicitly need it. I am not saying you need to live under a rock to have your private life stay private. What I am saying is before you give information to a large company, question if they really need it. If they don't need it, then in most places, it's not illegal to feed them fake information and still enjoy the benefits of their service. It's also not illegal to have multiple phone numbers or email addresses to give out so you are not tied to one specific identity. You could also go overboard with this and populate a ton of fake information into these companies to drown out the truthful information. It all depends on how far you want to go and how much time you have to waste. I will warn you to exercise caution when providing fake information to anyone. First ask yourself this question: is it illegal if I provide fake information to this individual/business/government? If you do not know the answer, do not provide fake information. It's better to stay out of trouble than to keep your privacy.

Gone are the days where a private life was possible by default. To achieve the same privacy before the Internet, we must work much harder for it. The amount of work can be very different depending on your situation. I have not even touched on the massive amount of ways your personal information is collected, nor would I be able to in anything smaller than a dictionary, but I hope to have given you a small taste. I hope it encourages you to step out of the predefined box that online advertising and marketing have put you in and reclaim your privacy. These companies are not paying you anything, so why should they be making money off your data? Besides the fact, what happens when one of these companies gets hacked and all your personal information is posted all over the Internet? You may not think of where your personal information is very often and you may not care, but after reading this article I hope you at least understand a little about the industry behind selling [you]. Be proactive, not reactive with the security of your personal information and you won't have a chance to regret it.

About the Author

Jacob Alexander is the Information Technology Manager/Information Security Specialist at Touchpoint Int'l Development Group, Inc. (https://www.tpidg.us). For more information on this topic, you can contact me at admin@tpidg.us



MAGAZINE BSD

Editor in Chief:
Ewa Dudzic

ewa@bsdmag.org

www.bsdmag.org

Contributing:

Teppei Fukuda and Kota Kanbe, Sitkowski, Jacob Alexander, Jean-Baptiste Boric, Rafael Santiago, Andrey Ferriyan, Natalia Portillo, E.G Nadhan, Daniel Cialdella Converti, Vitaly Repin, Henrik Nyh, Renan Dias, Rob Somerville, Hubert Feyrer, Kalin Staykov, Manuel Daza, Abdorrahman Homaei, Amit Chugh, Mohamed Farag, Bob Cromwell, David Rodriguez, Carlos Antonio Neira Bustos, Antonio Francesco Gentile, Randy Remirez, Vishal Lambe, Mikhail Zakharov, Pedro Giffuni, David Carlier, Albert Hui, Marcus Shmitt, Aryeh Friedman

Top Betatesters & Proofreaders:

Daniel Cialdella Converti, Eric De La Cruz Lugo, Daniel LaFlamme, Steven Wierckx, Denise Ebery, Eric Geissinger, Luca Ferrari, Imad Soltani, Olaoluwa Omokanwaye, Radjis Mahangoe, Katherine Dizon, Natalie Fahey, and Mark VonFange.

Special Thanks:

Denise Ebery

Katherine Dizon

Senior Consultant/Publisher:

Paweł Marciniak

Publisher:

Hakin9 Media SK,
02-676 Warsaw, Poland
Postepu 17D, Poland
worldwide publishing
editors@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

Processing Data in Parallel Using Multithreading

Real life being what it is, despite the isolation provided by the partitioned stack, a situation may still arise, where we might need to prevent more than one thread from executing a given function, or another block of code. For instance, we may be in the process of assembling a linked list, and it would be counter-productive to have two threads adding an element to the end of the list at the same time.

To cater for such eventualities, the thread library comes equipped with a useful gadget called a Mutex -–which is an abbreviation for ‘mutually exclusive switch’.

The above notwithstanding, a useful rule-of-thumb is: if you need to use a Mutex, you’ve done it wrong. The risks associated with adding a Mutex, which are outlined below, together with the fact that a Mutex negates any advantage of multi-threading, make its use highly undesirable.

We declare Mutexes globally since, for obvious reasons, they need to be thread-visible.

```
pthread_mutex_t xmutex;
```

Thereafter, we have to put in some code to lock the Mutex, which warrants a more detailed discussion.

The whole question of Mutex protection is analogous to that of file locking. Everything has to be done by agreement, and all parties wishing to access the resource have to cooperate. Accordingly, the correct way to do this, is as follows:

- As early as possible, initialise and unlock the Mutex
- Just before the protected block of code, attempt to lock the Mutex
- If this fails, it means that another thread got there first, so sleep and retry until it succeeds
- If it succeeds, lock the Mutex and execute the block of code
- Just after the protected block of code, unlock the Mutex.

This model makes several assumptions:

- It is impossible for two threads to lock the Mutex simultaneously.
- It is impossible for two threads to fail to lock the Mutex simultaneously.

These assumptions are correct in 99.999% of cases. There are cases when they are incorrect, a condition known as ‘thread deadlock’.

We initialise a Mutex to the unlocked condition, using the `mutex_init` function:

```
pthread_mutex_init(pthread_mutex_t *mutex,
pthread_mutexattr_t *attr);;
```

The variable ‘attr is a pointer to an attribute structure, which contains one member, which is a pointer to an int. Since we are not interested in changing this, let us pass in NULL, to get the default Mutex conditions.

```
if(pthread_mutex_init(&xmutex, NULL) ==
-1){

    printf(“Failed to initialise
mutex\n”);

}
```

We now have an initialised Mutex, and can protect our precious code. There are two calls available to do this. We can either

- Explicitly attempt to lock it, and check the return value:

```
if(pthread_mutex_lock(&xmutex) == -1){
    printf("Failed to lock mutex\n");
}
```

- We can call a function which does this for us:

```
if(pthread_mutex_trylock(&xmutex) == -1){
    printf("Failed to lock mutex\n");
}
```

If we want to wait for the code to become available, we will need a loop, which we traverse, just before the block of code:

```
    while(pthread_mutex_trylock(&mutex) == -1){
        sleep(1);
    }
```

Then, at the end of our protected block of code, we add the following lines of code:

```
if(mutex_unlock(&xmutex) == -1){
    printf("Failed to unlock mutex\n");
}
```

Now we're ready to try it. using our example code, above.

We will add a Mutex to the code, to force the threads to execute it in their order of arrival.

As it happens, this will not affect the overall timing, since the operation performed by each thread is sleep(). However, this generalisation does not apply to other functions. If we were performing real computationally intensive tasks, in our function, the advantages of multi-threading would be totally negated, as the threads would have to queue up to execute the function.

```
#include <stdio.h>
#include <ctype.h>
#include <pthread.h>
#define __REENTRANT
void *func(void *);
pthread_mutex_t mutex;
main(argc, argv)                    /* main */
int argc;
char **argv;
{
pthread_t thr[100];
int delay;
int i;
if(argc < 2){
        delay = 10;
    } else {
        delay = atoi(argv[1]);
    }
if(pthread_mutex_init(&mutex, NULL) == -1){
        printf("Failed to initialise mutex\n");
    }
for(i = 0; i < 100; i++){
        if((pthread_create(&thr[i], NULL, func,  (void *)&delay)) != 0){
            printf("Failed to create thr[%d]\n",i);
        }
    }
    for(i = 0; i < 100; i++){
        if(pthread_join(thr[i], NULL) != 0){
            printf("Failed to start thr[%d]\n",i);
        }
    }
    printf("All threads terminated\n");
}                                   /* main */
void *
func(delay)                         /* func */
void *delay;
{
    while(pthread_mutex_trylock(&mutex) == -1){
        sleep(1);
    }
```

```
    printf("Starting thread %d for %d
sex..\n", pthread_self(), *(int *)delay);

    sleep(*(int *)delay);

    printf("Thread %d returning..\n",
pthread_self());

    if(pthread_mutex_unlock(&mutex) ==
-1){
        printf("Can't unlock mutex\n");
    }
}                               /* func */
```

Performance Note

So, we experience the trouble of parallelising our application, we create a hundred threads, with no Mutexes, and sit back and watch it run.

Since we have one hundred parallel paths of execution, our process will run one hundred times faster, right?

The answer is, yes it will, but our application probably will not..

The process, within its process slot, will certainly run one hundred times faster, but that's not the whole story. The next question we should ask is, 'How often does the process run?'

If our process is the only one on the machine, then the application will certainly complete its operations at a considerably faster rate. However, in reality, we will be competing for CPU time with several dozen of other processes, and will have to take turns at running.

If our one hundred threads were one hundred processes, the overall application would, indeed, run nearly one 100 times faster. If our application is performance critical, then there is no choice, but to do it that way.

However, performance is not always the sole criterion. We should be concerned with the architectural elegance, or overall simplicity as well.

There are many situations, where each thread must read and write global data within the parent process. The overhead of synchronising communication with many children, and shunting data back and forth, may not be worth the improvement in performance.

Additionally, there may be computational reasons for using multi-threading, simply because we need to perform some operations in parallel. The simple case, where we leave the main thread listening for the next connection, while subsidiary threads handle each current connection is a good example of a threaded server.

About the Author

Mark Sitkowski is a Chartered Engineer, and a Corporate Member of the Institution of Electrical Engineers in London. His early career revolved around the writing of analog and digital circuit simulators and digital signal processing applications.

In Australia, he moved to writing financial software for the major banks, and telecommunications software for Telcos, together with conducting training courses on Unix and database applications. Formerly a consultant to Forticode Security, he currently works with Design Simulation Systems on mobile multi-factor authentication systems.

Design Simulation Systems Ltd
http://www.designsim.com.au
xmarks@exemail.com.au

Conventions over Restrictions – Programming the Python Way

BY MIKE MÜLLER

Python is a powerful programming language. It supports procedural and object-oriented programming, and provides important features for functional programming. Its dynamic typing and many meta-programming features allow doing nearly everything at run time. While this freedom provides lots of opportunities for elegant solutions, Python might also be perceived as a dangerously unrestricted language. The common solution is to use conventions to solve a problem in the preferred, hence, “pythonic” way. Experienced Python programmers tend to stick to these conventions as much as possible and only diverge when the benefits are substantial compared to that of the conventional solution.

This article gives a short overview of Python features focusing on conventions and their benefits. Since Python syntax is often called executable pseudo code, a reader with solid programming experience does not need to have previous knowledge of Python to follow along.

Interview with David Mytton

History

David Mytton was looking for a simple way to get metrics and alerts from his servers. He did not need a large enterprise product, and did not want to deal with legacy on-premise technologies. There was no product that fit the bill, so with the help of his school-friend, Harry, they decided to build their own.

In Short

David Mytton has been programming in Python and PHP for over 10 years, it was one of the earliest production MongoDB users (founding the London MongoDB User Group) He co-founded the Open Rights Group and can often be found cycling in London or drinking tea in Japan.

Can you tell our readers about yourself and your role nowadays?

I'm the Co-Founder & CEO at Server Density. Having built the original version of the product, I used to do a lot of coding but nowadays, I spend most of my time with the overall management of the business in financial planning, product strategy, technical and architectural oversight, commercial engagement with prospects and customers, hiring, PR & marketing and board or investor relations.

How did you get into programming?

I bought a book on ASP when I was thirteen and learned by building an e-commerce website for a local business. I moved onto PHP which allowed me to venture in web development appropriately. Starting Server Density was originally a project to learn Python since it's the main programming language we use at the company.

While having a wide field of expertise, which area do you emphasis on, and why?

My role is quite varied so there is no real emphasis on one thing. The two things I spend most of my time on are the product and the commercial side of the business.

With the product, we're up against some very well-funded competitors whereas, we fund ourselves therefore, most of our growth comes from customer revenue hence, we have to make tough decisions about what we build and when.

The commercial aspect of the business is tricky because, we have to be very disciplined in where we spend and who we hire. Server Density is mostly self service sales via credit card but, we are seeing majority of the customers asking for demos and a proper sales process.

Thus, we've been developing our approach to this and have even started hiring real sales people!

Being the founder of Server Density Limited, what was its inspiration? What was its purpose?

Back in 2009 I was working with another business and got frustrated at the lack of quality products to help us get alerts and graphs for our own server monitoring (the open source options like Nagios took too much time to configure and require a lot of effort to maintain and scale).

There was no service I could just buy and have it all work out of the box, so I decided to build it.

Who are your target group?

We have over 700 customers with the largest group (35%) from the US and only 15% from UK where we are based. These include the UK NHS Ambulance emergency response service, eCommerce stores such as Firebox through to Silicon Valley startups. Engineers are our usual users - anyone who has a server to manage or website to monitor the uptime of can use Server Density.

What was the most difficult and challenging implementation you've done so far? Please explain?

The biggest thing we did recently was to completely rebuild our time series data storage backend. Since the old custom system based on MongoDB had grown with us for 7 years but, was beginning to cost a lot in terms of money for infrastructure and time for maintenance. Therefore, we swapped it out for OpenTSB running on top of Google Cloud Bigtable. This gave us linear scalability of throughput and cost.

The challenges were to replace an existing system with a new database and also new schema on a new host provider with no downtime and migrate many TB of data without customer impact. And we did it!

What tools should be used oftenly and what for?

We're migrating all of our infrastructure over to Google Cloud and are making use of Kubernetes (running on Container Engine) and Terraform. Our existing infrastructure uses micro services but not containers, so we're taking the opportunity to containerize everything. Simply because, these tools are allowing us to fully automate and template our entire infrastructure - nothing is being done manually!

Do you have any specific goals for the rest of this year?

We have a few upcoming releases which will be taking advantage of some of the machine learning technologies available to us through Google Cloud. AI and machine learning are basically buzzwords at this point, so they have very little meaning for most people. Customers need not to worry that a product is using "machine learning" - it should be about the outcomes achieved with the technology, not technology just for technology's sake! We have eight years of monitoring and alerting data and soon we have some interesting things coming from that .

What's the best advice you can give to programmers?

Learn about infrastructure and how to run your apps. Any developer working in a modern organisation should currently be able to run their applications, not just pass it over to some separate ops team. This means monitoring, deployment, builds, testing, all under your control. Infrastructure technology is moving very fast and is fascinating to work with.

The problem with this is getting into the on-call mindset. You should be expected to be responsible for running your code in production but that does not mean you should sacrifice your quality of life.

Last year, we started a community called HumanOps to encourage discussion about this topic. Things like designing on-call rotas, understanding stress and introducing empathy into communications. Humans are an important part of any system and they are too easily forgotten. Check out http://www.humanops.com as there may be a meetup near you!.

Thank you

FREE READING

SDJOURNAL

Using Python Fabric to AGNU/Linux Server Configuration Tasks

What you will learn…

In this article, we will run through the basics of using Fabric to run local and remote commands and transfer files between servers. Through some examples, we will try to show some of its capabilities and use cases.

What you should know…

You are not required to have any prior experience to follow this article. However, if you are used to performing SSH system administration or application deployment tasks, it will be easier to realize some practical use cases.

About the author...

Renato Candido is a free (as in freedom) {software, hardware, and culture} enthusiast, who works as a technology consultant at Liria Technology, Brazil. He tries to solve technical's problems using these sorts of tools (he thinks the world would be a little better if all resources were free, as in freedom). He is an electronics engineer and enjoys learning things related to signal processing and computer science (and he thinks that there will be self-driving cars and speaking robots designed exclusively with free resources). Contact the author on http://www.renatocandido.org

www.sdjournal.org

COLUMN

The recent tragedy and fire at Grenfell Tower in London that claimed more than 80 lives may seem as far from IT and technology as it is possible to imagine. But to the engineering mind, it highlights the ongoing spectre of systemic failure and risk, where cascading error culminates in a disaster.

authored by Rob Somerville

It might seem insensitive at this early stage to equate one of the most devastating events to occur on British soil for decades with the cold hard reality and unforgiveness of complex systems, but I believe the IT industry and other sectors heavily dependent on technology, unlike the custodians and guardians of Grenfell Towers, have not ignored the many Cassandra like voices that have attempted to raise the alarm. In the case of Grenfell Towers, the residents repeatedly raised concerns about fire safety and the competency of the management group prior to the inferno. Indeed, the whole issue of the safety and viability of tower blocks in the UK has been an open secret amongst journalists, researchers, building engineers and many others for decades. For those with the eyes to see, it was only a matter of time.

When I first heard the news, I had that horrible sinking feeling in my guts, not least because of the scale of what happened, but because it was clear that for such a hellish event to have occurred, multiple dominoes would have to have fallen. Concrete doesn't burn. Tower blocks are inherently safe, provided you stick to the rules. And that includes understanding the engineering and scientific principles that the original (Hopefully competent designer) built the building or indeed the system upon.

Once you turn your back on these, you have ventured in the gambling territory. You might make a fortune, or you could find yourself in an ever increasing spiral of pain. Reality has a nasty habit of sneaking up behind you when hubris, over confidence and arrogance, takes root. Irrespective if you are a human, a cultural and political system or indeed a robot.

All it needs is for an over-abundance of trust for cracks to show. Like a deadly virus, the original trojan horse, this rot manifests itself as complacency, a complicity in a group think that everything will be OK provided we have a bit of paper and a box ticked. Ultimately, so much of the tragedy of life will be subscribed to human failure, as the universe works quite well without us. Technology and engineering is one of the greatest blessings bestowed on man, and as such, we need to devote to it respect rather than worship and thereby be seduced by the political and financial baggage that accompanies such power.

Good engineers, like good technologists and IT folk, are like magicians, conjuring something out of nothing. All we ask for is resource, and we can do the seemingly impossible. Yet the system failed regardless. In the face of Grenfell, we are akin to hired guns, those that sell our souls to the highest bidder, willing to say or do anything just for some cash. By

refusing to confront bad design decisions or poor policy, take a moral stance and say no, we may protect our careers and wallets but expose our bad conscience. We have become a part of the problem rather than the solution.

From a human level, both sides – the engineers, the creatives and senior management – live in very different worlds with very different perspectives and goals. So often there is an impasse, as neither side either wishes to, or actually can, understand. There is a connection though between the lofty heights of the plush executive boardroom and the dull gray work cubes in the trenches.

That connection is fear. Planet X, financial collapse, death, whatever. There is a tacit acceptance not to approach the elephant in the room as merely mentioning him brings with it the chance of a stampede. If you do mention it, censure, ridicule, and accusations of your professional competence or sanity are raised - if not to your face, behind your back. Both sides refuse to speak the truth, as the consequences are too much to bear. We can't fix the system properly as we grew too much, didn't invest when we should have. We didn't argue the risks hard enough. It takes a brave man or woman to openly declare the nakedness of the emperor.

I don't have an axe to grind, I see both sides. I'd just rather be in the trenches, coding all the time. Life has taught me otherwise. You need to walk barefoot amongst the fiery coals of the decision makers to truly understand the complexity and scale of the problem, not just of the technological system but the human one as well.

Once you get into the arena of risk management, your integrity and indeed your soul are on the line. Paper may cover the cracks, but lawsuits from the relatives of dead people delivered through your letterbox are a different matter. Those with such responsibilities tend to be the serious folk. They may even admit in their quieter moments that they too are scared. Sometimes, they are just there as a token gesture to pretend everything is OK.

We need to work together. Be totally honest, on both sides. If not, the battle between those with understanding and power will continue in ever decreasing circles until the arguments over policy, strategy and who is right ceases. By then, it will be too late.

You are responsible. Your code is important. What is more serious though, is your rigor in terms of ethical and moral adversity. Would lives depend on this? Livelihoods? What effect will this have on the next generation? Will you say No? Put your career on it? Whistle-blow?

In light of what happened on the 14th of June, we need a reality check, at all levels.

This article is dedicated to all at Grenfell in the hope that such devastation may never occur ever again, and that the truth, like sunlight, be found to be the best disinfectant.

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

Dr.WEB®
CureIt!™

EMERGENCY CURING

for Windows workstations and servers

including those running other anti-virus software

FUNCTIONS:

- Cures Windows workstations and servers.
- Verifies the quality of the anti-virus software currently in use.

FEATURES:

- Dr.Web CureIt! doesn't require installation and doesn't conflict with any known anti-virus; consequently there is no need to disable the anti-virus currently in use to check a system with Dr.Web CureIt!.
- Improved self-protection and an enhanced mode for more efficient countermeasures against Windows blockers.
- Dr.Web CureIt! is updated at least once an hour.
- The utility can be launched from removable media including USB storage devices.

LICENSING FEATURES:

The utility is available for free when used for non-business purposes.