BSD MAGAZINE

FOR NOVICE AND ADVANCED USERS

DEBUGGING APPLICATIONS
LLVM AND SANITIZERS IN BSD

PRACTICAL ZFS ON FREEBSD

C PROGRAMMING, UNIX AND MAIN DATA STRUCTURES

MONITORING OPENBSD
USING COLLECTD, INFLUXDB, AND GRAFANA

VOL 12 NO 06
ISSUE 06/2018 (106)
ISSN 1898-9144

IS AFFORDABLE FLASH STORAGE OUT OF REACH?

NOT ANYMORE!

IXSYSTEMS DELIVERS A FLASH ARRAY FOR UNDER $10,000.

Introducing FreeNAS® Certified Flash: A high performance all-flash array at the cost of spinning disk.

- Unifies NAS, SAN, and object storage to support multiple workloads
- Runs FreeNAS, the world's #1 software-defined storage solution
- Performance-oriented design provides maximum throughput/IOPs and lowest latency
- OpenZFS ensures data integrity
- Perfectly suited for Virtualization, Databases, Analytics, HPC, and M&E
- 10TB of all-flash storage for less than $10,000
- Maximizes ROI via high-density SSD technology and inline data reduction
- Scales to 100TB in a 2U form factor

The all-flash datacenter is now within reach. Deploy a FreeNAS Certified Flash array today from iXsystems and take advantage of all the benefits flash delivers.

Call or click today! **1-855-GREP-4-IX** (US) **| 1-408-943-4100** (Non-US) **| www.iXsystems.com/FreeNAS-certified-servers**

DON'T DEPEND ON CONSUMER-GRADE STORAGE.

KEEP YOUR DATA SAFE!

Buy from amazon.com

USE AN ENTERPRISE-GRADE STORAGE SYSTEM FROM IXSYSTEMS INSTEAD.

The FreeNAS Mini: Plug it in and boot it up — it just works.

- Runs FreeNAS, the world's #1 software-defined storage solution
- Unifies NAS, SAN, and object storage to support multiple workloads
- Encrypt data at rest or in flight using an 8-Core 2.4GHz Intel® Atom® processor
- OpenZFS ensures data integrity
- A 4-bay or 8-bay desktop storage array that scales to 48TB and packs a wallop
- Backed by a 1 year parts and labor warranty, and supported by the Silicon Valley team that designed and built it
- Perfectly suited for SoHo/SMB workloads like backups, replication, and file sharing
- Lowers storage TCO through its use of enterprise-class hardware, ECC RAM, optional flash, white-glove support, and enterprise hard drives

And really — why would you trust storage from anyone else?

iXsystems™

Call or click today! **1-855-GREP-4-IX** (US) | **1-408-943-4100** (Non-US) | **www.iXsystems.com/Freenas-Mini** or purchase on Amazon.

Editor's Word

MAGAZINE BSD

Dear Readers,

Tomorrow, June 1, 2018 marks a special day in Poland. It is Children's Day, which is celebrated in 86 countries worldwide at different times of the year. As we dedicate our time and energy to our little ones, I hope that the day adds more bliss and joyfulness to your life. Happy Children's Day!

Let's see what we have in this issue. For FreeBSD and NetBSD fans, we have two practical articles for you: one written by Abdorrahman Homaei and the second one by David Carlier. The first article, *Practical ZFS On FreeBSD*, will show you how amazing ZFS is. You will learn about ZFS design goals, how to enable ZFS on FreeBSD, and how to create your first ZFS Pool. You will also read about RaidZ, Snapshot and Rollback, and about Zpool Status. Additionally, you will learn how to share ZFS with NFS and how to monitor ZFS storage. The second article is about LLVM and Sanitizers. Don't feel left out if you are using a BSD OS other than FreeBSD. This article will also cover NetBSD, too. You will learn that LLVM provides the frontends and various tools, and the different types of sanitizers to help you with debugging applications. Moreover, we have published the first module of the Device Driver Development so I highly encourage you to enroll in this course and learn more from Rafael, the course instructor. For our Self Exposure section, Joel Carnat, an amazing blog creator, discusses how to monitor OpenBSD using Grafana, InfluxDB, and CollectD packages. Lastly, does our data lie safely with large Social Media corporations, and is data privacy a call for concern? Find the answer to these and more as you internalize Rob's column. Lastly, does our data lie largely with Social Media corporations and is data privacy a call for concern? Find the answer to these and more as you internalize Rob's column. Can corporations take steps to combat Unconscious Bias while interpreting such data? E.G.Nadhan expands on this in *Expert Speak*.

See you next time, and enjoy the issue!
Ewa & The BSD Team

P.S. Send me an email at ewa@bsdmag.org if you would like more information or would like to share your thoughts.

Table of Contents

MAGAZINE BSD

Editor in Chief:

Ewa Dudzic
ewa@bsdmag.org
www.bsdmag.org

Contributing:

Sanel Zukan, Luca Ferrari, José B. Alós, Carlos Klop, Eduardo Lavaque, 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

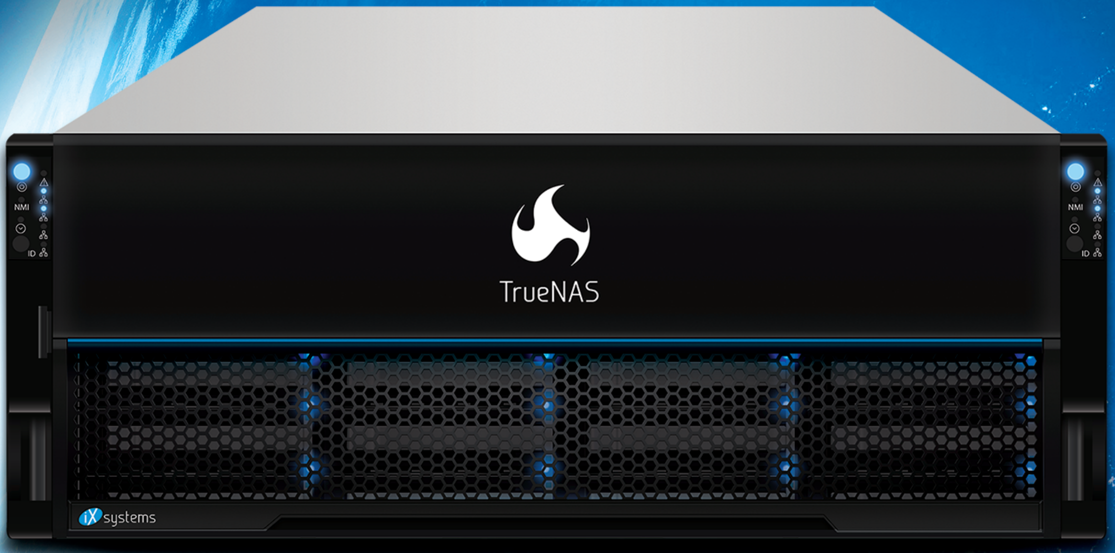
Publisher: Hakin9 Media SK,
02-676 Warsaw, Poland Postepu 17D, Poland
worldwide publishing
editors@bsdmag.org

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

Groundbreaking TrueNAS® M Series

intel XEON® SILVER inside™

DISRUPTING STORAGE AT WARP SPEED



LOWEST TCO SHARED STORAGE

Intel® Xeon® Scalable Family Processors

RESILIENT

- Self-healing Data
- Continuous Operation
- Easy Replication

WARP FACTOR IX

- Faster than SSD-based Hybrid Storage Arrays
- Flash-Turbocharged Data Access

EXPANDABLE

- Scales to 10PB using HGST Drives
- 10Gb/s-100Gb/s per NAS
- Non-disruptive Upgrades

COMPATIBLE

- Citrix, Veeam, & VMware Certified
- Unifies File, Block, & S3 Data
- Supports Leading Cloud Providers

Visit **iXsystems.com/TrueNAS** or call **(855) GREP-4-iX** today!

HGST
a Western Digital brand

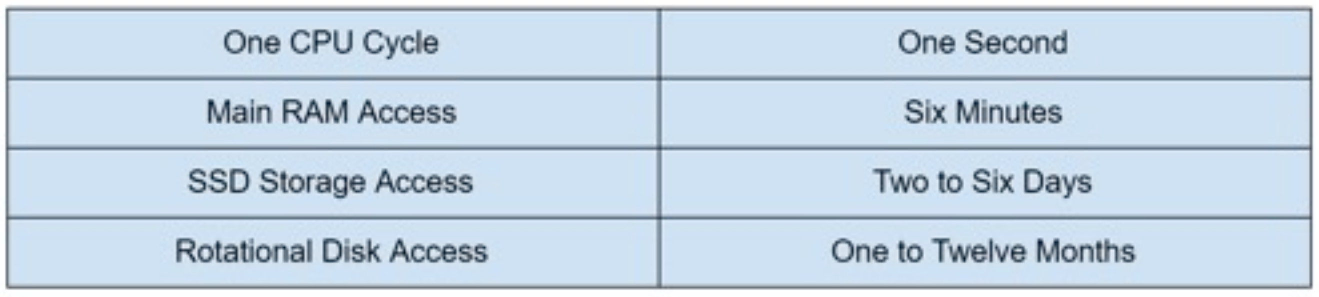
iX systems™

In Brief

Visualizing ZFS Performance

Many tools exist to understand ZFS performance challenges and opportunities, but a single table by renowned performance engineer Brendan Gregg will teach you to visualize the relationship between each tier of storage devices when architecting your TrueNAS or FreeNAS system.

Brendan Gregg worked closely with the ZFS Team at Sun Microsystems and later wrote the definitive book on Unix systems performance, *Systems Performance*. In the book, Brendan examines dozens of powerful performance analysis tools from top(1) to DTrace and plots his results with flame graphs to help establish baseline performance and pinpoint anomalies. I can't recommend the book enough and want to talk about a single chart in it that you might overlook. In the "Example Time Scale of System Latencies" on page 20, Brendan maps the latency of one CPU cycle to one second of time, and continues this mapping down through 14 more example elements of the computing stack. The resulting relative time scale ranges from one second for a CPU cycle to 32 *millennia* for a server to reboot. The four essential points in Brendan's scale for ZFS administrators are:



One CPU Cycle	One Second
Main RAM Access	Six Minutes
SSD Storage Access	Two to Six Days
Rotational Disk Access	One to Twelve Months

This deceptively simple chart provides the majority of what you need to understand ZFS performance challenges and opportunities. Newer flash-based storage devices like the NVDIMM and NVMe devices found in the new TrueNAS M-Series bridge the gap between SSDs and system RAM but the distinct performance tiers remain the same. Let's break them down:

One CPU Cycle

A CPU cycle is the one fixed point of reference for the performance of any given system, and most TrueNAS and FreeNAS systems maintain a surplus of CPU power. The operating system and services are the obvious primary consumers of this resource, but a ZFS-based storage system makes effective use of CPU resources in less obvious ways: checksumming, compressing, decompressing, and encrypting data. The data integrity guarantee made by ZFS is only possible thanks to a modern CPU's ability to calculate and validate data block checksums on the fly, a luxury not available on previous generations of systems. The CPU is also used for continuously compressing and decompressing data, reducing the burden on storage devices and yielding a performance gain.

Encryption performed by the CPU typically takes the form of SSH for network transfers or on-disk data block encryption. Faster SSH encryption improves network performance during replication transfers while data encryption can place an equal, if not greater burden on the storage system than compression. In all cases, CPU-based acceleration of compression, decompression, and encryption allows storage devices to perform at their best thanks to the *optimization* of the data provided to them.

Main RAM Access

Like the CPU, computer memory is not only used by the operating system and services, but it also provides a *volatile* form of storage that plays a key role in ZFS performance. Computer RAM is considered volatile because its contents are lost when the computer is switched off. While RAM performs slower than the CPU, it is also faster than all forms of *persistent* storage. ZFS uses RAM for its Adaptive Replacement Cache (ARC), which is essentially an intelligent read cache. Any data residing in the ARC, and thus RAM, is available *faster than any persistent storage device* can provide, at any cost. While ZFS is famous for aggressively using RAM, it is doing so for a good reason. Investing in RAM can be the greatest investment you can make for read performance.

SSD Storage Access

Sitting squarely between RAM and spinning disks in terms of performance are SSDs, now joined by the yet-faster NVMe cards and memory-class devices like NVDIMMs. Flash-based devices introduce *persistent* storage but generally pale in comparison to RAM for raw speed. With these stark differences in performance come stark differences in capacity and price, enlightening us to the fact that a high-performance yet cost-competitive storage stack is a compromise made of several types of storage devices. This has been termed “hybrid” storage by the industry. In practice, SSDs are the only practical foundation for an “all-flash array” for the majority of users and, like the ARC, they can also supplement slower storage devices. An SSD or NVMe card is often used for a ZFS *separate log device*, or SLOG, to boost the performance of synchronized writes, such as over NFS or with a database. The result is “all-flash” write performance and the data is quickly offloaded to spinning disks to take advantage of their capacity. Because this offloading takes place every five seconds by default, a little bit of SLOG storage goes a long way.

On the read side, a *level two* ARC, or L2ARC, is typically an SSD or NVMe-based read cache that can easily be larger than computer memory of the same price. Serving data from a flash device will clearly be faster than from a spinning disk, but slower than from RAM. Note that using an L2ARC does not mean you cut back on your computer memory too dramatically because the L2ARC index along with various ZFS metadata are still kept in RAM.

Rotational Disk Access

Finally, we reach the spinning disk. While high in capacity, disks are astonishingly slow in performance when compared to persistent and volatile flash and RAM-based storage. It is tempting to scoff at the relative performance of hard disks, but their low cost per *terabyte* guarantees their role as the heavy lifters of the storage industry for the foreseeable future. Stanley Kubrick's HAL 9000 computer in the movie 2001 correctly predicted that the future of storage is a bunch of adjacent chips, but we are a *long way* from that era. Understanding the relative performance of RAM, flash, and rotating disks will help you choose the right storage components for your ZFS storage array. The highly-knowledgeable sales team at iXsystems is here to help you quickly turn all of this theory into a budget for the storage system you need.

Michael Dexter

Senior Analyst

Source: https://www.ixsystems.com/blog/

BSDCan - The BSD Conference

BSDCan, a BSD conference held in Ottawa, Canada, quickly established itself as the technical conference for people working on and with 4.4BSD based operating systems and related projects. The organizers have found a fantastic formula that appeals to a wide range of people from extreme novices to advanced developers.
Tutorials: 6-7 June 2018 (Wed/Thu)
Conference: 8-9 June 2018 (Fri/Sat)
Location
University of Ottawa, in the DMS (Desmarais) building.

Source: https://www.bsdcan.org/2018/

EuroBSDcon 2018

University Politehnica of Bucharest, Bucharest, Romania
20 - 23 September, 2018
EuroBSDcon is the European annual technical conference gathering users and developers working on and with 4.4BSD (Berkeley Software Distribution) based operating systems family and related projects. EuroBSDcon gives the exceptional opportunity to learn about latest news from the BSD world, witness contemporary deployment case studies, and meet personally other users and companies using BSD oriented technologies. EuroBSDcon is also a boilerplate for ideas, discussions and information exchange, which often turn into programming projects. The conference has always attracted active programmers, administrators and aspiring students, as well as IT companies at large, which found the conference a convenient and quality training option for its staff. We firmly believe that high profile

education is vital to the future of technology, and hence greatly welcome students and young people to this regular meeting.

Source: https://2018.eurobsdcon.org/

pfSense 2.4.3-RELEASE-p1 and 2.3.5-RELEASE-p2 Available

The release of pfSense® software versions 2.4.3-p1 and 2.3.5-p2, now available for upgrades!
pfSense software versions 2.4.3-p1 and 2.3.5-p2 are maintenance releases bringing security patches and stability fixes for issues present in the pfSense 2.4.3 and 2.3.5-p1 releases.
This release includes several important security patches, including the issues discussed last week:

FreeBSD Security Advisory for CVE-2018-8897

FreeBSD-SA-18:06.debugreg

FreeBSD Errata Notice for CVE-2018-6920 and CVE-2018-6921

FreeBSD-EN-18:05.mem

Fixed a potential LFI in pkg_mgr_install.php #8485 pfSense-SA-18_04.webgui

Fixed a potential XSS in pkg_mgr_install.php #8486 pfSense-SA-18_05.webgui

Fixed a potential XSS vector in RRD error output encoding #8269 pfSense-SA-18_01.packages

Fixed a potential XSS vector in diag_system_activity.php output encoding #8300 pfSense-SA-18_02.webgui

Changed sshd to use delayed compression #8245

Added encoding for firewall schedule range descriptions #8259

Aside from security updates, the new versions include a handful of beneficial bug fixes for various minor issues.

Upgrading to pfSense 2.3.5-RELEASE-p2

Updating from an earlier pfSense 2.3.x release to pfSense 2.3.5-p2 on an amd64 installation that could otherwise use pfSense 2.4.x requires configuring the firewall to stay on pfSense 2.3.x releases as follows:

Navigate to System > Update, Update Settings tab
Set Branch to Legacy stable version (Security / Errata Only 2.3.x)
Navigate back to the Update tab to see the latest pfSense 2.3.x update
The same change is required to see pfSense 2.3.x packages for users staying on pfSense 2.3.x.
Firewalls running 32-bit (i386) installations of pfSense software do not need to take any special actions to remain on 2.3.x as they are unable to run later versions.

Update Troubleshooting

If the update system offers an upgrade to pfSense but the upgrade does not proceed, ensure that the firewall is set to the correct update branch as mentioned above. If the firewall is on the correct branch, refresh the repository configuration and upgrade the script by running the following commands from the console or shell:

```
pkg-static clean -ay; pkg-static install -fy pkg pfSense-repo pfSense-upgrade
```

In some cases, the repository information may need to be rewritten. This can be accomplished by switching to a development branch, checking for updates, and then switching back to the appropriate branch and checking for updates again.

Reporting Issues

This release is ready for a production use. Should any issues come up with pfSense 2.4.3-RELEASE-p1 or 2.3.5-RELEASE-p2, please post about them on the the forum, the mailing list, or on the /r/pfSense subreddit.

Source:
https://www.netgate.com/blog/pfsense-2-4-3-release-p1-and-2-3-5-release-p2-now-available.html

ServerU

Rack-mount networking server

Designed for BSD and Linux Systems

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

Practical ZFS On FreeBSD

What Is ZFS?

ZFS is an advanced file system that originally developed by Sun. ZFS Combining the roles of volume manager and file system with unique advantages. ZFS is aware of the underlying structure of the disks and can detect low-level interrupt and provides RAID mechanism. ZFS is capable of share its volume separately. ZFS's awareness of the physical layout of the disks let you grow your storage without any hassle. ZFS also has a number of different properties that can be applied to each file system, giving many advantages to creating a number of different file systems and datasets rather than a single monolithic file system.

Lately, ZFS development has moved to the OpenZFS Project.

ZFS Design Goals

ZFS has three major design goals:

- Data integrity: All data includes a checksum of the data. When data is written, the checksum is calculated and written along with it. When that data is later read back, the checksum is calculated again. If the checksums do not match, a data error has been detected. ZFS will attempt to automatically correct errors when data redundancy is available.

- Pooled storage: physical storage devices are added to a pool, and storage space is allocated from that shared pool. Space is available to all file system and can be increased by adding new storage devices to the pool.

- Performance: multiple caching mechanisms provide increased performance. ARC is an advanced memory-based read cache. The second level of disk-based read cache can be added with L2ARC, and disk-based synchronous write cache is available with ZIL.

Enable ZFS On FreeBSD

FreeBSD supports ZFS natively and all you need to do is to add this line to "`/etc/rc.conf`" manually:

```
 zfs_enable="YES"
```

```
Or with:
```

```
# echo 'zfs_enable="YES"' >> /etc/rc.conf
```

```
Then start the service:
```

```
# service zfs start
```

A minimum of 4GB of RAM is required for comfortable usage, but individual workloads can vary widely.

Create First ZFS Pool

ZFS can work directly with device node but you can also create your own disk with truncate:

```
# truncate -s 2G disk_1

# truncate -s 2G disk_2

# truncate -s 2G disk_3

# truncate -s 2G disk_4
```

Then create your own pool and name it storage:

```
# zpool create storage /root/disk_1
/root/disk_2 /root/disk_3 /root/disk_4

# zpool list
```

As you can see we have 7.94G storage. This pool is not taking advantage of any ZFS features. To create a dataset on this pool with compression enabled:

`Compression Property`

```
# zfs create storage/myfolder

# zfs set compression=gzip storage/myfolder
```

It is now possible to see the data and space utilization by issuing df:

```
storage             7.7G     23K    7.7G
0%    /storage

storage/myfolder    7.7G     23K    7.7G
0%    /storage/myfolder
```

you can disable compression by:

```
# zfs set compression=off storage/myfolder
```

`Copies Property`

If you have something important you can keep more copies of it:

```
# zfs create storage/archive

# zfs set copies=2 storage/archive
```

To destroy the file systems and then destroy the pool as it is no longer needed:

```
# zfs destroy storage/myfolder

# zfs destroy storage/archive

# zpool destroy storage

zpool set autoexpand=on mypool
```

RaidZ, Snapshot, and Rollback

A variation on RAID-5 that allows for better distribution of parity and eliminates the "RAID-5" write hole (in which data and parity become inconsistent after a power loss). Data and parity are striped across all disks within a raidz group.

Try creating a file system snapshot which can be rolled back later:

```
# zfs snapshot storage/myfolder@now
```

You can restore to the created snapshot with:

```
# zfs rollback storage/myfolder@now
```

Also, you can list all ZFS datasets and snapshots:

```
# zfs list -t all
```

Zpool Status

A pool's health status is described by one of three states:

- online (all devices operating normally)
- degraded (one or more devices have failed, but the data is still available due to a redundant configuration)
- faulted (corrupted metadata, or one or more faulted devices, and insufficient replicas to continue functioning)

You can get pool status by:

```
# zpool status
```

Hot Spares

ZFS allows devices to be associated with pools as "hot spares". These devices are not actively used in the pool, but when an active device fails, it is automatically replaced by a hot spare. To create a pool with hot spares, specify a "spare" vdev with any number of devices.

In the example, we have raidz consist of 4 disks and 1 backup disk.

```
# zpool create storage raidz /root/disk_1
/root/disk_2 /root/disk_3 /root/disk_4
spare /root/disk_5
```

Share ZFS With NFS

ZFS supports NFS natively and you can share pools in a network.

Add these lines to “/etc/rc.conf”:

```
rpcbind_enable="YES"

mountd_flags="-n"

nfs_server_enable="YES"

mountd_enable="YES"

Then issue this command:

zfs set sharenfs=on storage/myfolder

showmount command will list NFS export
list:

# showmount -e
```

Monitoring ZFS Storage

With ZFS built-in monitoring system you can view pool I/O statistics in real time. It shows the amount of free and used space in the pool, read and write operations per second and I/O band-width.

By issuing this command status will be shown every 1 second:

```
# zpool iostat 1
```

Conclusion

ZFS Combining the roles of volume manager and file system with unique advantages. It's aware of the underlying structure of the disks and can detect low-level interrupt and provides RAID mechanism.

Useful Links

https://www.freebsd.org/doc/handbook/zfs.html

https://docs.oracle.com/cd/E23824_01/html/821-1448/gayne.html

https://blogs.oracle.com/roch/nfs-and-zfs,-a-fine-combination

https://www.freebsd.org/doc/handbook/zfs-term.html

https://www.freebsd.org/doc/handbook/zfs-zpool.html

https://www.freebsd.org/doc/en/books/faq/all-about-zfs.html

Meet 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 started his own company (etesal amne sara Tehran) in Feb, 2017. His company based in Iran silicon valley.

Full CV: ***http://in4bsd.com***

His company: ***http://corebox.ir***

BSD Certification

The BSD Certification Group Inc. (BSDCG) is a non-profit organization committed to creating and maintaining a global certification standard for system administration on BSD based operating systems.

WHAT CERTIFICATIONS ARE AVAILABLE?

BSDA: Entry-level certification suited for candidates with a general Unix background and at least six months of experience with BSD systems.

BSDP: Advanced certification for senior system administrators with at least three years of experience on BSD systems. Successful BSDP candidates are able to demonstrate strong to expert skills in BSD Unix system administration.

WHERE CAN I GET CERTIFIED?

We're pleased to announce that after 7 months of negotiations and the work required to make the exam available in a computer based format, that the BSDA exam is now available at several hundred testing centers around the world. Paper based BSDA exams cost $75 USD. Computer based BSDA exams cost $150 USD. The price of the BSDP exams are yet to be determined.

Payments are made through our registration website:
https://register.bsdcertification.org//register/payment

WHERE CAN I GET MORE INFORMATION?

More information and links to our mailing lists, LinkedIn groups, and Facebook group are available at our website:
http://www.bsdcertification.org

Registration for upcoming exam events is available at our registration website:
https://register.bsdcertification.org//register/get-a-bsdcg-id

LLVM and Sanitizers in BSD

LLVM and clang frontend is available on various BSD as the main compiler for FreeBSD x86, ppc and arm since the 10.x (fully was optional in the previous 9.x branch), OpenBSD x86 and arm since 6.2, NetBSD x86, arm, ppc and sparc64. LLVM provides the frontends and various tools, and on the other side of the spectrum, there are different types of sanitizers to help with debugging applications.

What you will learn:

What are the available sanitizers and tools

Their various availability and working state for each BSD.

What you need to know:

Basic knowledge of LLVM usage with any frontend

Experience in debugging with language using LLVM infrastructure

LLVM is mainly used via its frontends to generate LLVM bytecode, which is eventually compiled to native binary format. It also comes with (optional) a set of tools from static code analysis, code formatter (clang-format), LLVM IR “interpreter” (lli), LLVM bytecode quality measuring (llvm-mca) to the sanitizers suite (a subset of is used by gcc), which we are going to focus in this article.

The sanitizers are capable of detecting bugs at runtime that are not predictable when compiling. What if a buffer has a constant size but the program allows writing from user entry without size checking?

Address Sanitizer

This sanitizer (aka asan) detects memory usage error at run-time, dangling pointers usage or buffer boundaries issues to summarize. The flag to pass is `-fsanitize=address`.

For a basic example:

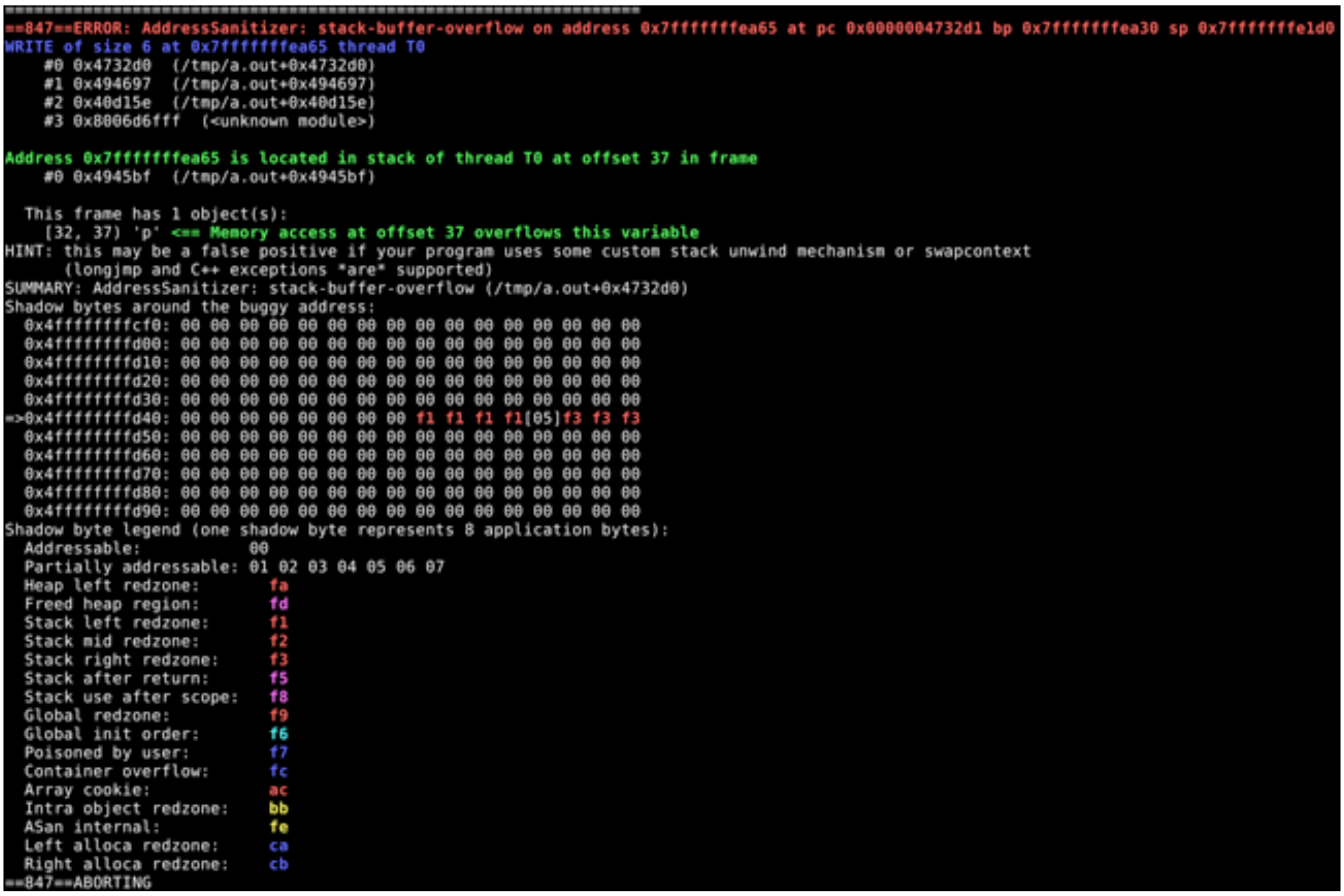
```
#include <string.h>

int main(void)

{

  char p[5] = {};

  strcpy(p, "12345");

  return 0;

}
```

The above code will generate the report which is shown on Figure 1.

We can see the fifth character, ‘5’ out of the boundaries.

And this code:

```
#include <string.h>

#include <stdio.h>

#include <stdlib.h>

void test(char **ptr)

{
```



```
=================================================================
==847==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7fffffffea65 at pc 0x0000004732d1 bp 0x7fffffffea30 sp 0x7fffffffe1d0
WRITE of size 6 at 0x7fffffffea65 thread T0
    #0 0x4732d0  (/tmp/a.out+0x4732d0)
    #1 0x494697  (/tmp/a.out+0x494697)
    #2 0x40d15e  (/tmp/a.out+0x40d15e)
    #3 0x8006d6fff  (<unknown module>)

Address 0x7fffffffea65 is located in stack of thread T0 at offset 37 in frame
    #0 0x4945bf  (/tmp/a.out+0x4945bf)

  This frame has 1 object(s):
    [32, 37) 'p' <== Memory access at offset 37 overflows this variable
HINT: this may be a false positive if your program uses some custom stack unwind mechanism or swapcontext
      (longjmp and C++ exceptions *are* supported)
SUMMARY: AddressSanitizer: stack-buffer-overflow (/tmp/a.out+0x4732d0)
Shadow bytes around the buggy address:
  0x4ffffffffcf0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x4ffffffffd00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x4ffffffffd10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x4ffffffffd20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x4ffffffffd30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x4ffffffffd40: 00 00 00 00 00 00 00 00 f1 f1 f1 f1[05]f3 f3 f3
  0x4ffffffffd50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x4ffffffffd60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x4ffffffffd70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x4ffffffffd80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x4ffffffffd90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
==847==ABORTING
```

Figure 1. The report

```
  char *p = malloc(5);

  strcpy(p, "abcd");

  free(p);

  *ptr = p;

}


int main(void)

{

  char *p;

  test(&p);

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

  return 0;

}
```

gives the output which can be seen on Figure 2.

Where, we see the attempt to use the 5 bytes allocated and freed earlier

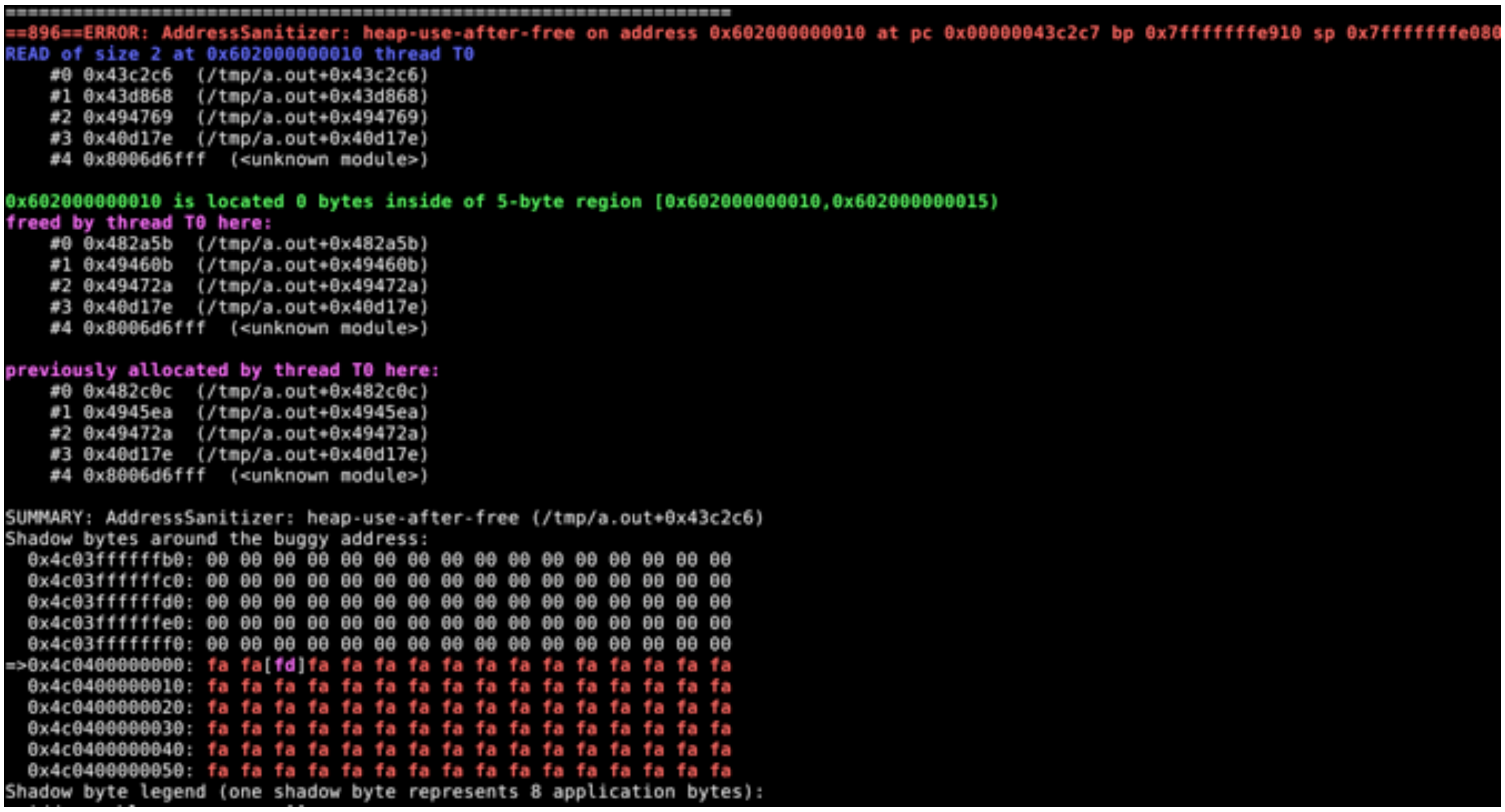
Supported by: FreeBSD and NetBSD

Memory Sanitizer

This sanitizer (aka msan) is mainly used to detect uninitialized values when attempted to be used.

For example: this code

```
#include <stdio.h>

#include <stdlib.h>


int main(int argc, char **argv)

{

  int *arr = (int *)malloc(sizeof(*arr) *
10);
```



```
=================================================================
==896==ERROR: AddressSanitizer: heap-use-after-free on address 0x602000000010 at pc 0x00000043c2c7 bp 0x7fffffffe910 sp 0x7fffffffe080
READ of size 2 at 0x602000000010 thread T0
    #0 0x43c2c6  (/tmp/a.out+0x43c2c6)
    #1 0x43d868  (/tmp/a.out+0x43d868)
    #2 0x494769  (/tmp/a.out+0x494769)
    #3 0x40d17e  (/tmp/a.out+0x40d17e)
    #4 0x8006d6fff  (<unknown module>)

0x602000000010 is located 0 bytes inside of 5-byte region [0x602000000010,0x602000000015)
freed by thread T0 here:
    #0 0x482a5b  (/tmp/a.out+0x482a5b)
    #1 0x49460b  (/tmp/a.out+0x49460b)
    #2 0x49472a  (/tmp/a.out+0x49472a)
    #3 0x40d17e  (/tmp/a.out+0x40d17e)
    #4 0x8006d6fff  (<unknown module>)

previously allocated by thread T0 here:
    #0 0x482c0c  (/tmp/a.out+0x482c0c)
    #1 0x4945ea  (/tmp/a.out+0x4945ea)
    #2 0x49472a  (/tmp/a.out+0x49472a)
    #3 0x40d17e  (/tmp/a.out+0x40d17e)
    #4 0x8006d6fff  (<unknown module>)

SUMMARY: AddressSanitizer: heap-use-after-free (/tmp/a.out+0x43c2c6)
Shadow bytes around the buggy address:
  0x4c03ffffffb0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x4c03ffffffc0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x4c03ffffffd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x4c03ffffffe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x4c03fffffff0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x4c0400000000: fa fa[fd]fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x4c0400000010: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x4c0400000020: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x4c0400000030: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x4c0400000040: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x4c0400000050: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
```

Figure 2. The output

```
  arr[5] = 0;

  if (arr[argc])

    printf("%d\n", arr[argc]);

  free(arr);

  return 0;

}
```

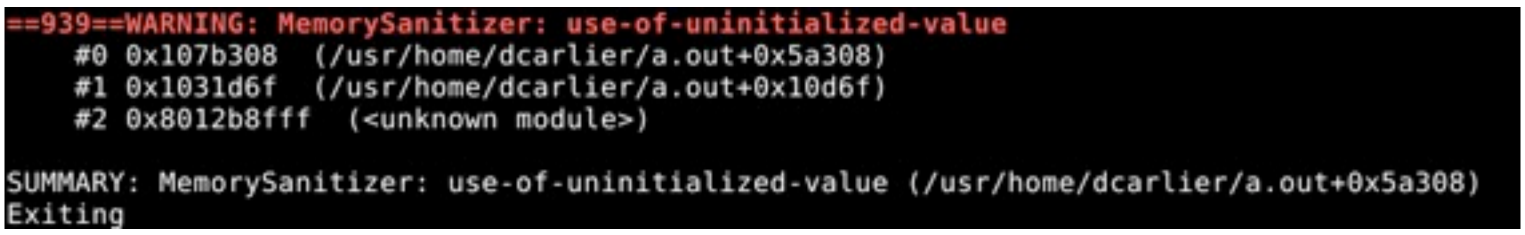
will give an output (see Figure 3) highlighting the use of the uninitialized array item.

Supported by: FreeBSD (from clang 7) and NetBSD

Thread Sanitizer

This sanitizer (aka tsan) is mainly used to detect race conditions in multi-thread context, which is a usually quite edgy sort of bugs to solve. The impact in terms of performance is more noticeable than the rest of the sanitizers. However, it’s delicate to use it in production code.

```
#include <pthread.h>

#include <stdio.h>


static int a = 12;

void *changeA(void *arg)

{

  a = *((int *)arg);

  printf("a is %d\n", a);

  return 0;

}


int main(int argc, char **argv)

{

  pthread_t pt[2];

  int c1 = 13;

  int c2 = 11;

  pthread_create(&pt[0], NULL, changeA, 
(void *)&c1);

  pthread_create(&pt[1], NULL, changeA, 
(void *)&c2);

  pthread_join(pt[1], NULL);

  pthread_join(pt[0], NULL);

  return 0;

}
```



```
==939==WARNING: MemorySanitizer: use-of-uninitialized-value
    #0 0x107b308  (/usr/home/dcarlier/a.out+0x5a308)
    #1 0x1031d6f  (/usr/home/dcarlier/a.out+0x10d6f)
    #2 0x8012b8fff  (<unknown module>)

SUMMARY: MemorySanitizer: use-of-uninitialized-value (/usr/home/dcarlier/a.out+0x5a308)
Exiting
```

Figure 3. The use of the uninitialized array

Again, this code would not cause visible issue. But with the sanitizer instrumentation, the data race with the global is detected. See Figure 4.

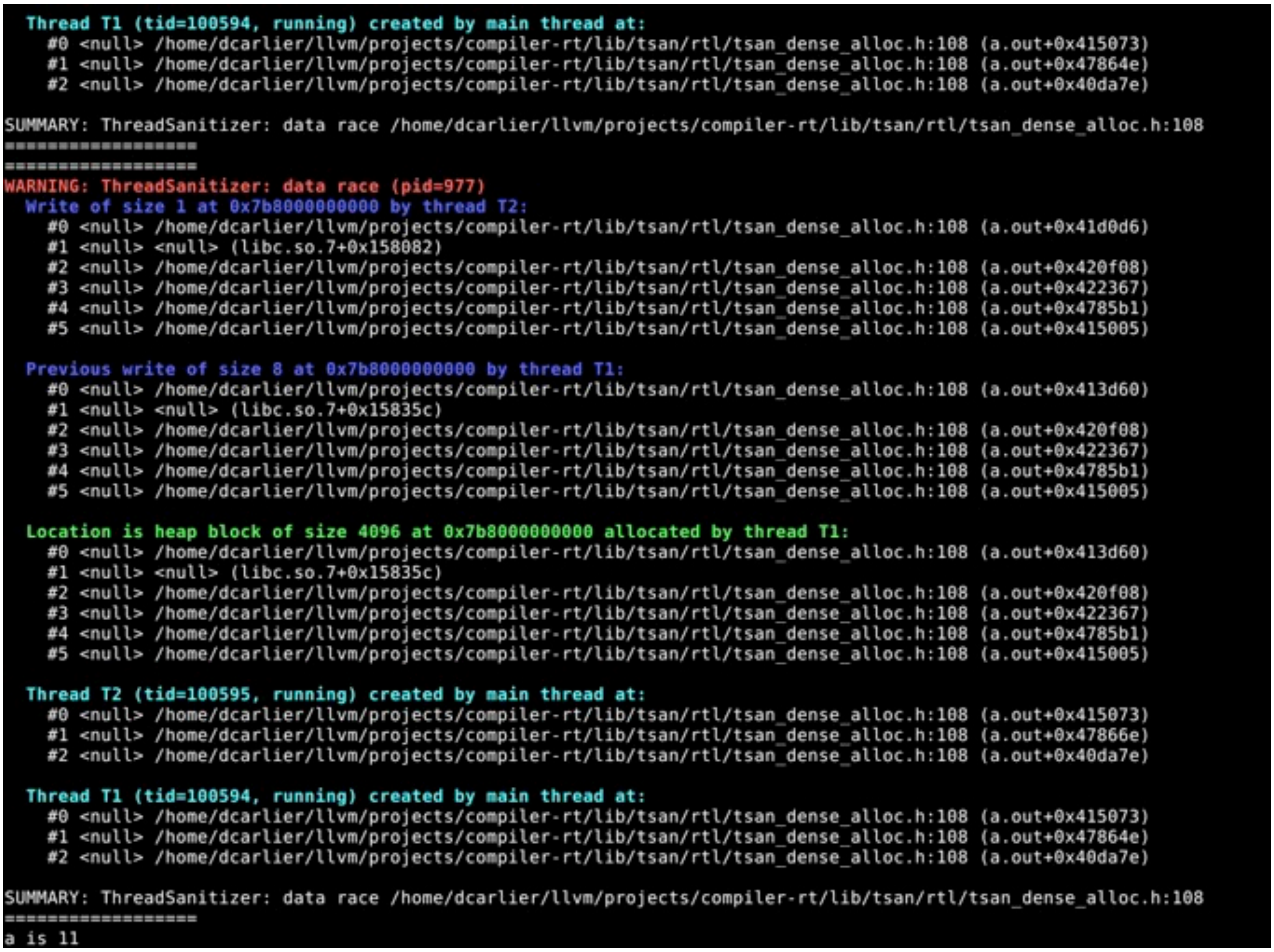
Supported by: FreeBSD and NetBSD

Undefined Behavior Sanitizer

The role of the Undefined Behavior Sanitizer (aka ubsan) is to detect subtle undefined behavior bugs as integer overflow, division by zero, and invalid bit shift operations (a typical case with signed types trying to shift bits as it was unsigned). Ubsan is often used in conjunction with other sanitizers like asan, msan or tsan.

For example, let's try a classic integer overflow:

```
#include <string.h>


int main(int argc, char *argv[])

{

  int r = 1 << 32;

  return 0;

}
```



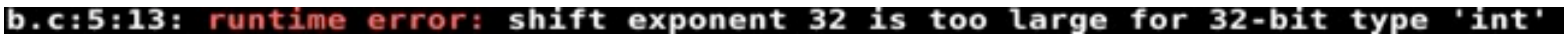
```
  Thread T1 (tid=100594, running) created by main thread at:
    #0 <null> /home/dcarlier/llvm/projects/compiler-rt/lib/tsan/rtl/tsan_dense_alloc.h:108 (a.out+0x415073)
    #1 <null> /home/dcarlier/llvm/projects/compiler-rt/lib/tsan/rtl/tsan_dense_alloc.h:108 (a.out+0x47864e)
    #2 <null> /home/dcarlier/llvm/projects/compiler-rt/lib/tsan/rtl/tsan_dense_alloc.h:108 (a.out+0x40da7e)

SUMMARY: ThreadSanitizer: data race /home/dcarlier/llvm/projects/compiler-rt/lib/tsan/rtl/tsan_dense_alloc.h:108
==================
==================
WARNING: ThreadSanitizer: data race (pid=977)
  Write of size 1 at 0x7b8000000000 by thread T2:
    #0 <null> /home/dcarlier/llvm/projects/compiler-rt/lib/tsan/rtl/tsan_dense_alloc.h:108 (a.out+0x41d0d6)
    #1 <null> <null> (libc.so.7+0x158082)
    #2 <null> /home/dcarlier/llvm/projects/compiler-rt/lib/tsan/rtl/tsan_dense_alloc.h:108 (a.out+0x420f08)
    #3 <null> /home/dcarlier/llvm/projects/compiler-rt/lib/tsan/rtl/tsan_dense_alloc.h:108 (a.out+0x422367)
    #4 <null> /home/dcarlier/llvm/projects/compiler-rt/lib/tsan/rtl/tsan_dense_alloc.h:108 (a.out+0x4785b1)
    #5 <null> /home/dcarlier/llvm/projects/compiler-rt/lib/tsan/rtl/tsan_dense_alloc.h:108 (a.out+0x415005)

  Previous write of size 8 at 0x7b8000000000 by thread T1:
    #0 <null> /home/dcarlier/llvm/projects/compiler-rt/lib/tsan/rtl/tsan_dense_alloc.h:108 (a.out+0x413d60)
    #1 <null> <null> (libc.so.7+0x15835c)
    #2 <null> /home/dcarlier/llvm/projects/compiler-rt/lib/tsan/rtl/tsan_dense_alloc.h:108 (a.out+0x420f08)
    #3 <null> /home/dcarlier/llvm/projects/compiler-rt/lib/tsan/rtl/tsan_dense_alloc.h:108 (a.out+0x422367)
    #4 <null> /home/dcarlier/llvm/projects/compiler-rt/lib/tsan/rtl/tsan_dense_alloc.h:108 (a.out+0x4785b1)
    #5 <null> /home/dcarlier/llvm/projects/compiler-rt/lib/tsan/rtl/tsan_dense_alloc.h:108 (a.out+0x415005)

  Location is heap block of size 4096 at 0x7b8000000000 allocated by thread T1:
    #0 <null> /home/dcarlier/llvm/projects/compiler-rt/lib/tsan/rtl/tsan_dense_alloc.h:108 (a.out+0x413d60)
    #1 <null> <null> (libc.so.7+0x15835c)
    #2 <null> /home/dcarlier/llvm/projects/compiler-rt/lib/tsan/rtl/tsan_dense_alloc.h:108 (a.out+0x420f08)
    #3 <null> /home/dcarlier/llvm/projects/compiler-rt/lib/tsan/rtl/tsan_dense_alloc.h:108 (a.out+0x422367)
    #4 <null> /home/dcarlier/llvm/projects/compiler-rt/lib/tsan/rtl/tsan_dense_alloc.h:108 (a.out+0x4785b1)
    #5 <null> /home/dcarlier/llvm/projects/compiler-rt/lib/tsan/rtl/tsan_dense_alloc.h:108 (a.out+0x415005)

  Thread T2 (tid=100595, running) created by main thread at:
    #0 <null> /home/dcarlier/llvm/projects/compiler-rt/lib/tsan/rtl/tsan_dense_alloc.h:108 (a.out+0x415073)
    #1 <null> /home/dcarlier/llvm/projects/compiler-rt/lib/tsan/rtl/tsan_dense_alloc.h:108 (a.out+0x47866e)
    #2 <null> /home/dcarlier/llvm/projects/compiler-rt/lib/tsan/rtl/tsan_dense_alloc.h:108 (a.out+0x40da7e)

  Thread T1 (tid=100594, running) created by main thread at:
    #0 <null> /home/dcarlier/llvm/projects/compiler-rt/lib/tsan/rtl/tsan_dense_alloc.h:108 (a.out+0x415073)
    #1 <null> /home/dcarlier/llvm/projects/compiler-rt/lib/tsan/rtl/tsan_dense_alloc.h:108 (a.out+0x47864e)
    #2 <null> /home/dcarlier/llvm/projects/compiler-rt/lib/tsan/rtl/tsan_dense_alloc.h:108 (a.out+0x40da7e)

SUMMARY: ThreadSanitizer: data race /home/dcarlier/llvm/projects/compiler-rt/lib/tsan/rtl/tsan_dense_alloc.h:108
==================
a is 11
```

Figure 4. The data race with the global

Which will give the following output with this generic flag `-fsanitize=undefined`. See Figure 5. Since it's not a dynamic value, modern compilers can detect such overflow. Another example of ubsan usage, for C++ only, is to check if the internal pointer to vtable of a given instance class really points to the right function pointers. For example, with the flag `-fsanitize=vptr`, this code which would not trigger any apparent fault in a normal situation,

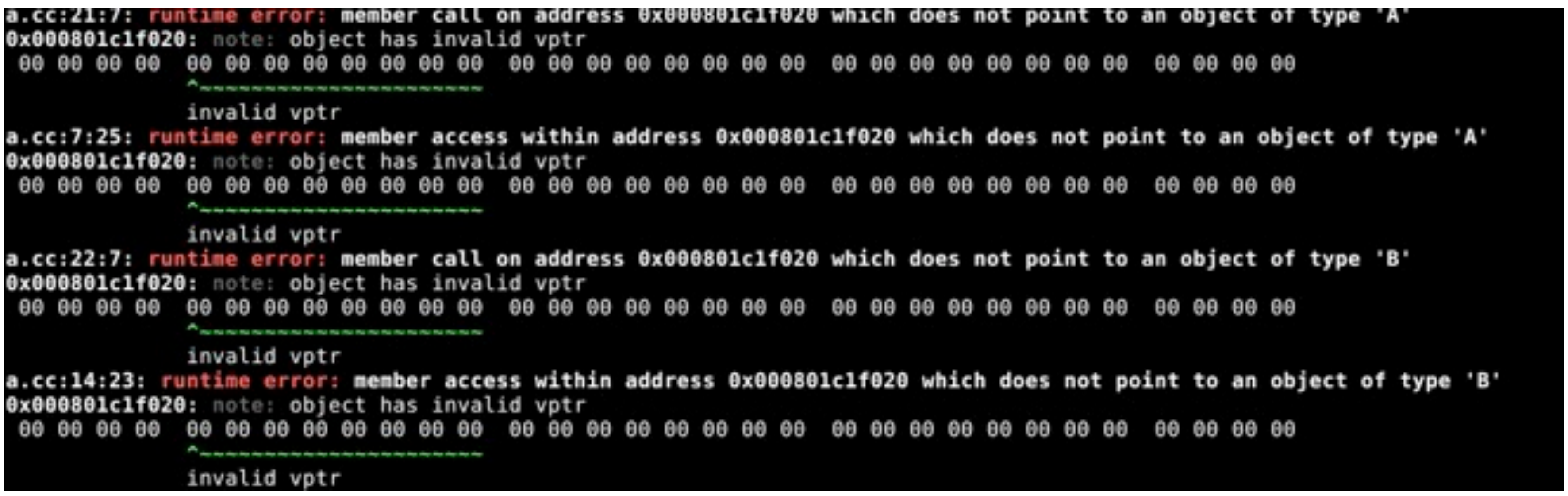
```
#include <string>

class A {

  char m[5];

  virtual void virt() = 0;

public:

  char *getM() { return m; }

};

class B : public A {

  int i;

  void virt() { i = 0; }

public:

  int getI() { return i; }

};



int main(void)

{

  unsigned char *p = new unsigned
char[sizeof(B)];

  B *pB = (B *)p;

  pB->getM();

  pB->getI();

  delete [] p;

  return 0;

}
```

will display the following output, where the allocated pointer is not a proper B class instance. See Figure 6.



```
b.c:5:13: runtime error: shift exponent 32 is too large for 32-bit type 'int'
```

Figure 5. The output with this generic flag `-fsanitize=undefined`



```
a.cc:21:7: runtime error: member call on address 0x000801c1f020 which does not point to an object of type 'A'
0x000801c1f020: note: object has invalid vptr
 00 00 00 00  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  00 00 00 00
              ^~~~~~~~~~~~~~~~~~~~~~~
              invalid vptr
a.cc:7:25: runtime error: member access within address 0x000801c1f020 which does not point to an object of type 'A'
0x000801c1f020: note: object has invalid vptr
 00 00 00 00  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  00 00 00 00
              ^~~~~~~~~~~~~~~~~~~~~~~
              invalid vptr
a.cc:22:7: runtime error: member call on address 0x000801c1f020 which does not point to an object of type 'B'
0x000801c1f020: note: object has invalid vptr
 00 00 00 00  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  00 00 00 00
              ^~~~~~~~~~~~~~~~~~~~~~~
              invalid vptr
a.cc:14:23: runtime error: member access within address 0x000801c1f020 which does not point to an object of type 'B'
0x000801c1f020: note: object has invalid vptr
 00 00 00 00  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  00 00 00 00
              ^~~~~~~~~~~~~~~~~~~~~~~
              invalid vptr
```

Figure 6. The output, where the allocated pointer is not a proper B class instance

Supported by: FreeBSD, OpenBSD (from clang 7) and NetBSD

Leak Sanitizer

As its name suggests, it detects memory/resource leaks.

Supported by: At the moment, only a NetBSD support is planned by the NetBSD foundation.

SafeStack

Safestacks protects the software against stack overflows without a noticeable performance hit. It is more useful for systems without such protection originally.

The flag to pass in order to enable it is:

`-fsanitize=safe-stack`

Therefore, a simple program that would function somehow in normal conditions

```
#include <string.h>


int main(int argc, char *argv[])

{

  char p[3];

  strcpy(p, "abcdefghi");

  return 0;

}
```

will simply provoke a segmentation fault.

Supported by: FreeBSD and NetBSD

X-ray instrumentation

This feature allows getting accurate function call tracing, giving the opportunity to inspect the bottlenecks without significant performance impact, and allowing itself to be used in production simultaneously.

With this code, we can use attributes to define instrumented or not instrumented functions to check, for example, the ones that are suspiciously the bottlenecks in terms of performance, and the ones we are sure are not.

```
#include <unistd.h>


static int global = 0;


void always_instrument(int)
__attribute__((xray_always_instrument));

void never_instrument(int)
__attribute__((xray_never_instrument));

void reset()
__attribute__((xray_never_instrument));


void always_instrument(int i)

{

  global += i;

  sleep(3);

}


void never_instrument(int i)

{

  global += i;

  sleep(3);

}
```

```
void reset()

{

  global = 0;

}



int main()

{

  for (int i = 0; i < 3; i ++) {

    always_instrument(i);

    never_instrument(i);

  }



  reset();



  return 0;

}
```

Here, we generate the trace of our application to check which part of the code count in the total spent (by default, the trace is not generated). See Figure 7.

Since our never_instrumented and reset functions are not instrumented on purpose, the delta with main (instrumented by default) appears clearly. See Figure 8.

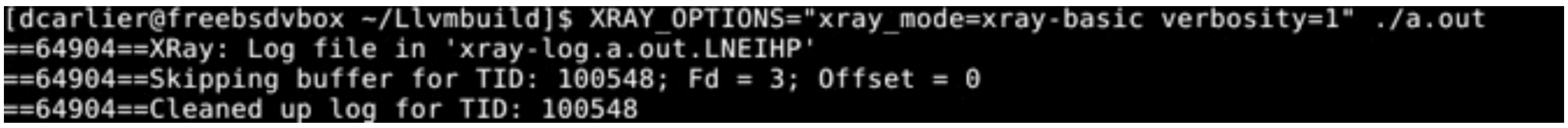
Supported by: FreeBSD (from clang 7), OpenBSD (from clang 7), and NetBSD

Fuzzer

Fuzzing, in general, is a very useful software testing technique based on giving random data (called corpus) to the software or library in question.

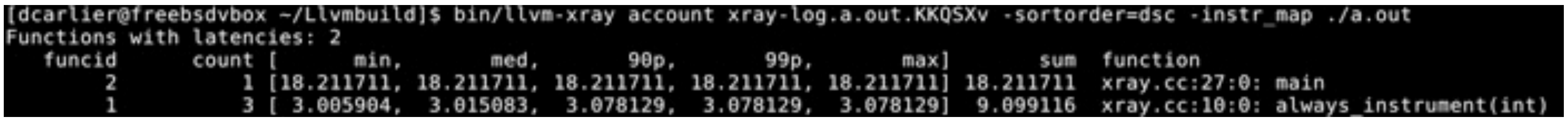
In the LLVM standpoint, there is a possibility to build a binary to be used for fuzzing. First, we need to define the LLVMTestFuzzerOneInput C function (`main` entry point is already defined) as:

```
int LLVMTEstFuzzerOneInput(uint8_t *input,
size_t inputlen)

{

     <exploiting input>

     return 0;

}
```



```
[dcarlier@freebsdvbox ~/Llvmbuild]$ XRAY_OPTIONS="xray_mode=xray-basic verbosity=1" ./a.out
==64904==XRay: Log file in 'xray-log.a.out.LNEIHP'
==64904==Skipping buffer for TID: 100548; Fd = 3; Offset = 0
==64904==Cleaned up log for TID: 100548
```

Figure 7. The trace is not generated



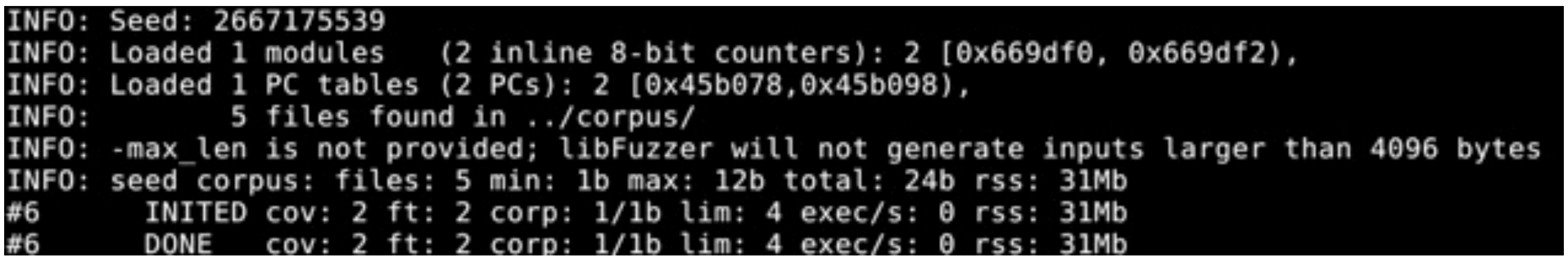
```
[dcarlier@freebsdvbox ~/Llvmbuild]$ bin/llvm-xray account xray-log.a.out.KKQSXv -sortorder=dsc -instr_map ./a.out
Functions with latencies: 2
   funcid      count [      min,       med,       90p,       99p,       max]       sum  function
        2          1 [18.211711, 18.211711, 18.211711, 18.211711, 18.211711] 18.211711  xray.cc:27:0: main
        1          3 [ 3.005904,  3.015083,  3.078129,  3.078129,  3.078129]  9.099116  xray.cc:10:0: always_instrument(int)
```

Figure 8. The delta with main (instrumented by default) appears clearly

Usually, libFuzzer is used in conjunction with a sanitizer to spot possible bugs, the ones we mentioned earlier, in the process. Therefore, as libFuzzer, ubsan, msan, asan and tsan support parallel jobs.

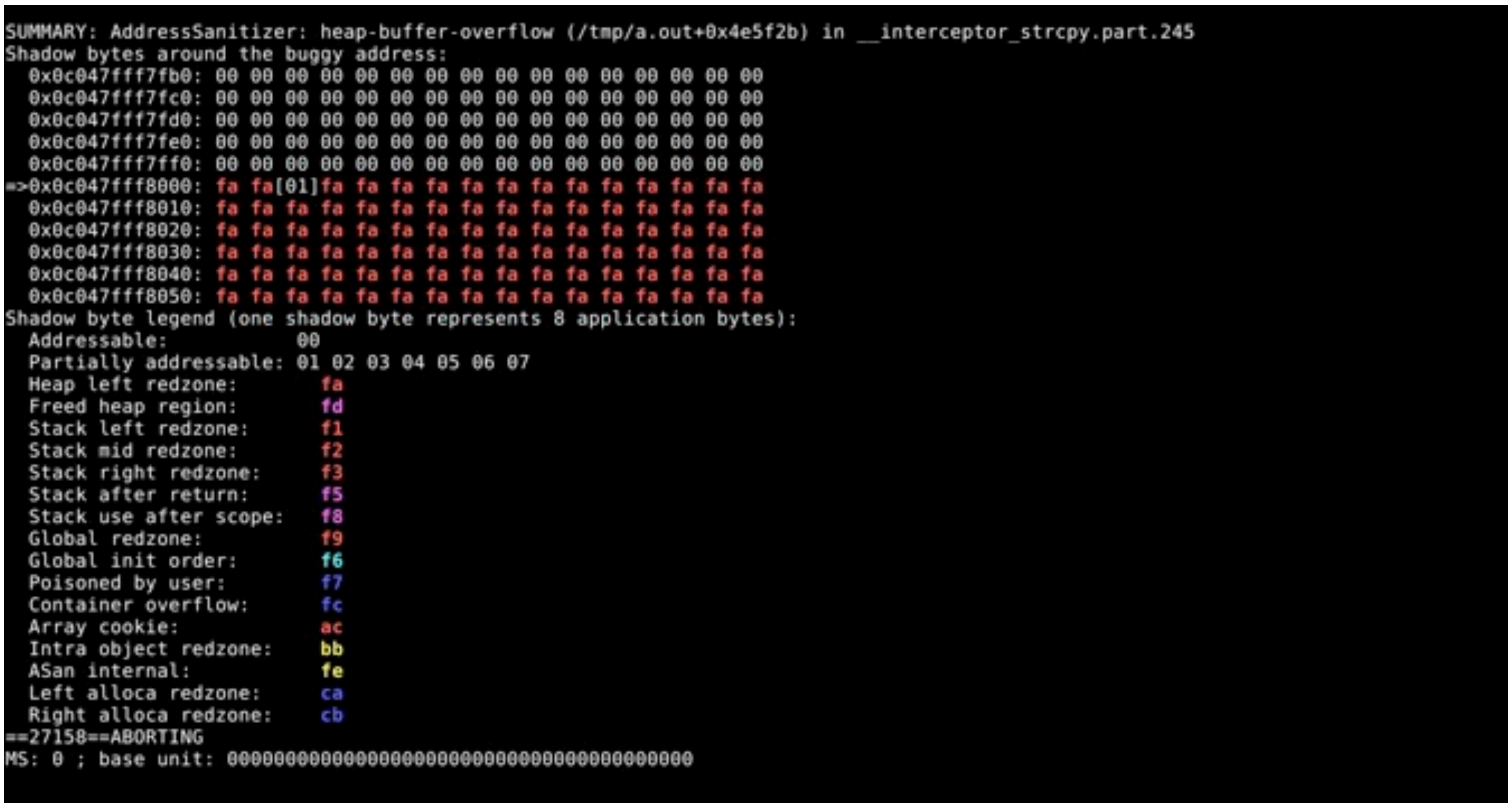
```
#include <sys/types.h>

#include <string.h>

#include <stdint.h>
```

```
void myLibraryCall(const char *);


extern "C" int
LLVMFuzzerTestOneInput(const uint8_t
*input, size_t inputlen)

{

  myLibraryCall((const char *)input);

  return 0;

}
```



```
INFO: Seed: 2667175539
INFO: Loaded 1 modules   (2 inline 8-bit counters): 2 [0x669df0, 0x669df2),
INFO: Loaded 1 PC tables (2 PCs): 2 [0x45b078,0x45b098),
INFO:        5 files found in ../corpus/
INFO: -max_len is not provided; libFuzzer will not generate inputs larger than 4096 bytes
INFO: seed corpus: files: 5 min: 1b max: 12b total: 24b rss: 31Mb
#6      INITED cov: 2 ft: 2 corp: 1/1b lim: 4 exec/s: 0 rss: 31Mb
#6      DONE   cov: 2 ft: 2 corp: 1/1b lim: 4 exec/s: 0 rss: 31Mb
```

Figure 9. A corpus data containing an element to trigger the buffer overflow



```
SUMMARY: AddressSanitizer: heap-buffer-overflow (/tmp/a.out+0x4e5f2b) in __interceptor_strcpy.part.245
Shadow bytes around the buggy address:
  0x0c047fff7fb0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c047fff7fc0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c047fff7fd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c047fff7fe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c047fff7ff0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0c047fff8000: fa fa[01]fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8010: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8020: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8030: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8040: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8050: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
==27158==ABORTING
MS: 0 ; base unit: 0000000000000000000000000000000000000000
```

Figure 10. A corpus data containing an element to trigger the buffer overflow

```
void myLibraryCall(const char *data)

{

  static char buf[8];

  strcpy(buf, data);

}
```

Let's compile this with these flags `-fsanitize=fuzzer,address`

So we have a corpus data containing an element to trigger the buffer overflow. See Figure 9 and 10.

Supported by: FreeBSD (from clang 7), OpenBSD (from clang 7) and NetBSD

Development

Various FreeBSD developers/contributors had done most of the work for FreeBSD. I personally ported libFuzzer, msan, and X-Ray instrumentation.

For NetBSD, mainly kamil from the NetBSD foundation.

For OpenBSD, I ported ubsan, libFuzzer, and X-ray instrumentation.

Conclusion

The sanitizers are definitely useful within a developer's toolset, whether it's for professional purpose or as BSD contribution to detecting subtle bugs exhaustively in the userland. The support varies; while the BSD, NetBSD, and FreeBSD support most of the features, OpenBSD only supports a subset of the features, but always under active development.

References

https://llvm.org/docs/GettingStarted.html

https://blog.netbsd.org/tnf/entry/the_llvm_sanitizers_stage_accomplished

Meet the Author

*David Carlier is an experienced developer and used to handle some languages like C/C++, Java, Python with Linux, *BSD and Win32 Operating Systems and worked inside startups and bigger companies as well. Personally a big fan of FreeBSD and OpenBSD. C/C++ are his preferred programming language most of the time. He writes and reviews articles for BSDMag http://www.bsdmag.org. He contributes modestly to OpenBSD ports and time in time to the source.*
He has been interviewed by BSDNow show http://www.bsdnow.tv/episodes/2017_10_18-software_is_storytelling. Also He did some small contributions for FreeBSD and DragonflyBSD operating system.

C Programming, UNIX and Main Data Structures

Nowadays, UNIX stands more as a model for an operating system to follow rather than an operating system implementation. In the beginning, UNIX as a software was originally written at Bell Labs by two famous developers: Kenneth Thompson and Dennis Ritchie.

In 1963, Bell Labs and others companies joined to create a new operating system with the following requirements:

- Multi-user
- Be multi-tasking
- Be capable of storing and sharing data and programs on a large scale
- Allow data sharing among users and groups

This project/system was called MULTICS. It was developed on a GE-645 but in 1969, MULTICS was still not a fully working operating system. Other companies have continued the MULTICS development but Bell Labs, not seeing much future in this project, decided to quit.

During that time, some developers within Bell Labs were frustrated with the decision to quit and, started developing a simpler MULTICS version themselves.

Actually, Kenneth Thompson was tired of trying to play 'Space Travel' in MULTICS. Space Travel was a simulation game he originally wrote for MULTICS. However, the operating system was not much able of well executing that game. Still in 1969, facing the MULTICS execution problems, Thompson decided to port Space Travel to another unused computer in his laboratory, a PDP-7. Having developed tons of workarounds to make possible the game execution, this porting effort quickly became an entire operating system, fully written in PDP-7 assembly.

In order to convince the company managers about how serious the project was, it was presented as a future text editor; later, it evolved to a general-purpose operating system, named UNIX.

Several technologies were developed as support for the UNIX project. The most important was the C programming language, developed by Dennis Ritchie.

C was developed as an evolution of Thompson's B language. The UNIX was totally re-written in C and it was released as a commercial operating system. C allowed UNIX to be developed in a more portable way with regard to other computer architectures. The portability introduced by the C language was a seminal step in computing field as a whole.

The name 'UNIX' is just a pun on the name 'MULTICS'.

The features and fruits of UNIX

The interesting thing about UNIX is that its source code was used in Operating System classes until the code was closed by AT&T. If you are interested in looking at the original UNIX source code developed by Thompson & Ritchie (UNIX V6), search for the book: "Lions' commentary on UNIX 6th Edition – with source code" (ISBN 1-57398-013-7). The book was organized by professor John Lions as a study aid for his students. The book has the complete UNIX V6 code listing and sections where Lions discusses all system parts, explaining the code.

Around the world, many operating system developers have debuted in this field reading those notes from professor Lions.

When UNIX was closed and its source code could not be used in classes anymore, another college professor, Andrew Stuart Tanenbaum, decided to create the MINIX project. MINIX would become one of the first efforts of creating a UNIX-like operating system without the original source code from Bell Labs. Also, one of the most famous books about Operating Systems widely used in many Computer Science classes until today originated from the MINIX project. The MINIX development is still active. Currently, MINIX is licensed under BSD.

The first BSD versions were branches of the original UNIX. At present, any "BSD" system can be considered a derivative of the original UNIX ancestor.

Linux is another famous modern UNIX-like but its development started in 90's, and it does not have any code from original UNIX.

User programs can be easily ported from UNIX-like to another. The sharing between all those UNIX-like systems is only possible because all of them follow an important document called Single Unix Specification (http://opengroup.org/unix). This document summarizes what an operating system must implement to be considered a UNIX-like.

The Single Unix Specification is composed of three documents: ANSI C, XPG4, and POSIX.

The ANSI C is about the standard for the C language implementation, including syntax, libraries and other features. The XPG4 is about standards for X server, the graphical UNIX interface. The POSIX document lists all system calls and signals that a UNIX-like must implement.

How internally a UNIX-like works does not matter for the Single Unix Specification, but how this UNIX-like externally reacts and replies is very important. It defines if the system can be considered a UNIX-like or not. Due to it, maybe POSIX could be considered the most important document in the whole standard.

The POSIX signals and system calls

The POSIX standard defines 31 signals. Those signals must be implemented by a UNIX-like. Table 1 lists more details about each of them. Signals are important because the system uses them to manage its processes. Processes are able to send and receive signals.

Table 1: The POSIX signals.

Portable Number	Name	Description
1	SIGHUP	Hangup
2	SIGINT	Terminal interrupt signal
3	SIGQUIT	Terminal quit signal
-	SIGILL	Illegal instruction
-	SIGTRAP	Trace/breakpoint trap
6	SIGABRT	Process abort signal
-	SIGIOT	Process abort signal. (PDP-11)
-	SIGEMT	Obsolete
-	SIGUNUSED	Unused
9	SIGKILL	Kill (can not be ignored)
-	SIGFPE	Erroneous arithmetic operation
-	SIGUSR1	User-defined signal 1
-	SIGBUS	Access to an undefined portion of a memory object
-	SIGSEGV	Invalid memory reference
-	SIGUSR2	User-defined signal 2
-	SIGPIPE	Write on a pipe with no one to read it (broken-pipe)
14	SIGALARM	Alarm clock
15	SIGTERM	Termination signal (can be ignored)
-	SIGCHLD	Child process terminated, stopped, or continued
-	SIGCONT	Continue executing, if stopped
-	SIGSTOP	Stop executing (cannot be caught or ignored)
-	SIGTSTP	Terminal stop signal
-	SIGTTIN	Background process attempting read
-	SIGTTOU	Background process attempting write

Opposing to the standard signals, there are several system calls. Some of them are related to the file system (open, read, and write), time utilities (gettimeoftheday), networking (send, recv, accept), etc.

As you should see, some system call names are used also to name functions inside the standard C library. These C library functions are directly related to those system calls but the standard C library only implements the user side of them. Being those C functions only tiny windows to the true system calls.

A brief overview of C language

C is a powerful procedural language created by Dennis Ritchie at Bell Labs to simplify the UNIX development. One of the most impressing concepts introduced by C language was the notion of portability.

Functionalities like the unary operator “sizeof” and the C pre-processor made the process of porting software easier than before.

Although at first glance it was developed as support technology for the UNIX project, the use of C has gone beyond the UNIX frontiers and C has been proved to be a well succeed programming language. Even nowadays, despite the advent of more modern and “friendly” programming languages, there are still tons of legacy and new software being maintained and developed in C.

The compilation process of a C program can be divided into 3 main phases: Pre-processing, Compiling and Linking.

The pre-processing phase is performed by the C pre-processor software. This program handles all defined macro stuff and the compiler directives.

The compiling phase is managed by the C compiler itself. It parses the source codes seeking to verify all data. Aborting when some lexical or syntax error are found.

The linking is done by the linker program. This software will properly generate the executable code.

The C language is surprisingly powerful and compact. The basic data types include int, long, short, char, float and double. It is also possible to use these types with unsigned values. In this case, the type name must be prefixed by the reserved word “unsigned”. It is also possible to use some extensions of some types such as “long long”, etc.

User-defined types are also allowed, in this case, the reserved word “typedef” is used for doing it.

One of the most powerful features in C language is pointers. Pointers, as the name suggest, instead of directly storing data, they indirect to the data: it does store a memory address where some data is in.

The memory management is all up to the developer. For beginners, it makes C a little bit hard. Anyway, programming in C is valuable for any student who really wants to know how their system works.

Since this course assumes C Programming skills as pre-requisite, more details about the core of the language will not be given.

Nonetheless, if you have problems with pointers, structs, how to implement classical data structures in C, basic flow control statements, ANSI C functions, function pointers, or a more advanced C macro usage, I would suggest you study more deeply on the C language before starting taking this course.

Main Data Structures

As any software, operating systems take advantage of many data structures to store and manipulate relevant data.

There two main data structures widely used: queues and stacks.

Queues

Queues store data in the way its name suggests: the current piece of data is stored after the last stored piece of data. When some data is needed to be removed, the older data is always removed first. Due to this behavior, lists are also known by the acronym FIFO (First In First Out).

Stacks

Stacks also handle data as its name suggests: the data is stacked. When some data is removed, the newer data is removed before the older. This behavior is explained by the acronym LIFO (Last In First Out).

Lists

Lists are a generalization of queues and stacks. There are many implementations of lists: singly linked, doubly linked. The data handling in this kind of data structure is more loose. New data can be added at any point in the list and the data can be removed irrespective of its position in the list.

Depending on the list type (singly, doubly linked), some operations will be easier than others. Anyway, this freedom of editing the data is achieved by taking advantages of pointers.

Usually, a list item is composed of a data payload and a pointer. The set of many items pointing to other items compose the whole list. Figure 1 illustrates the general idea of a list item.

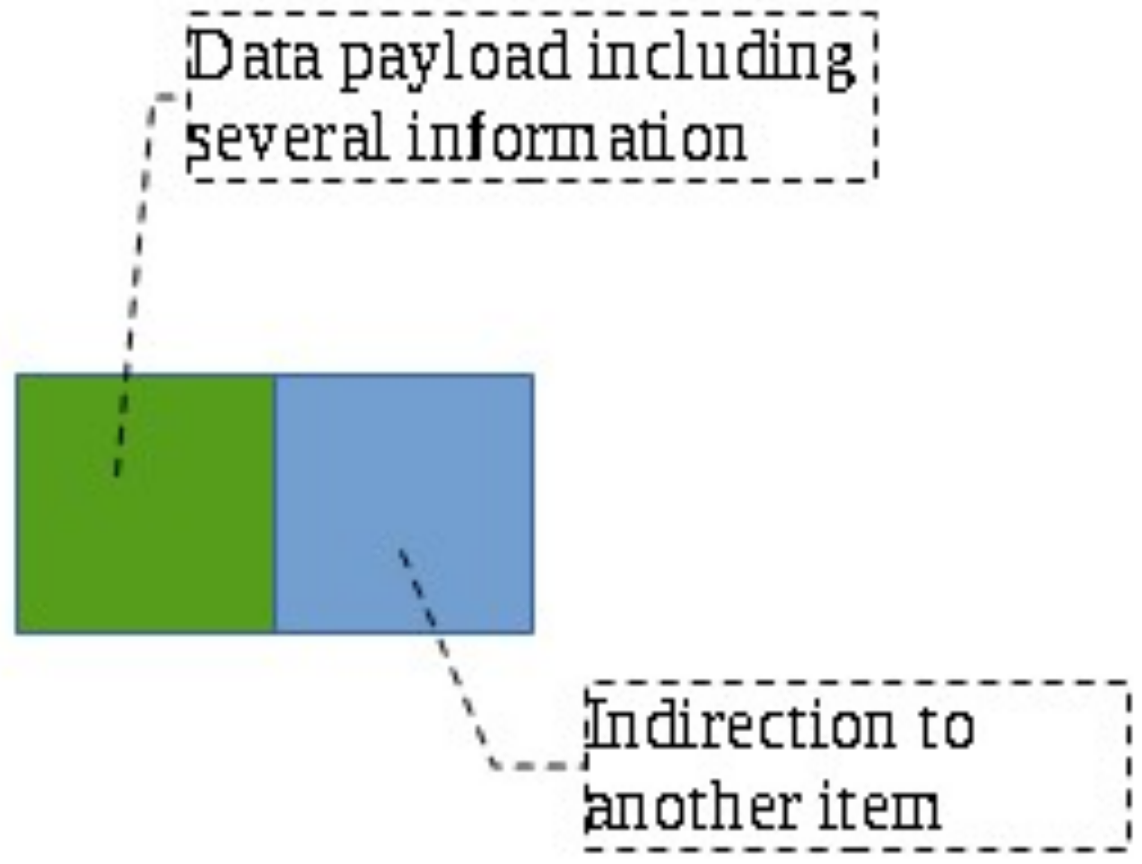

Data payload including several information
Indirection to another item

Figure 1. A list item

A singly linked list item just points to its next item. When there is no next item, the pointer is null. Figure 2 illustrates a singly linked list.

A doubly linked list item points to its next item but also points to its prior item. When there is no next or prior items, the related pointers are set to null. Figure 3 illustrates a doubly linked list.

Trees

Trees can be understood as a multi-level linked list. In this case, the data payload of the list item also counts with a sub-indirection used to point to several sub-lists with the same behavior adopted by the main list. In other words, it is just a mirrored structure. Usually, recursive algorithms should be implemented to handle data in a tree structure.

How those data structures can be expressed in C

Data structures can be expressed in C using the reserved word “struct”. Indirections can be implemented as pointers to the structure itself:

```
struct list_item {

  void *data;

  size_t data_size;

  struct list_item *next;

};
```

FreeBSD and NetBSD implement native utilities for list handling. To use those utilities is considered a best practice because the code is stable, optimized and mitigates the insertion of new bugs related to such basic operations.

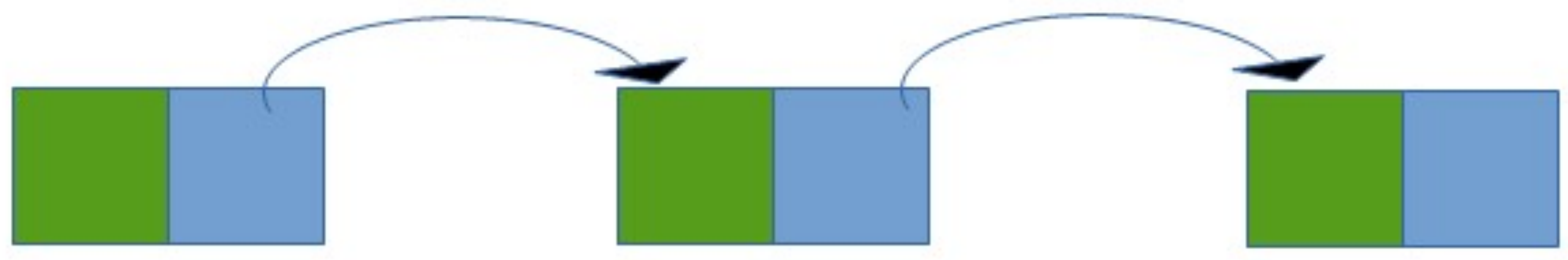


Figure 2. A single linked list

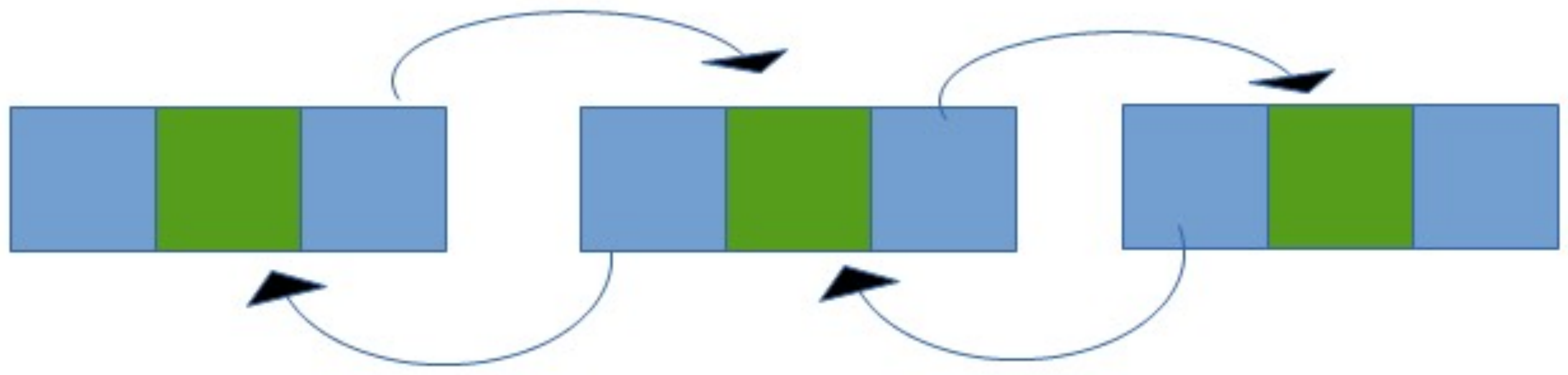


Figure 3. A doubly linked list

Conclusions

UNIX initially was a commercial operating system.

The C Language was primarily developed to make easier UNIX ports. However, the design behind C, which is focused on a compact programming language specification and a programming language that does not limit the developer, has made the C language a good choice for several real world software projects. C has a large influence on many modern programming languages too.

College teachers had been using UNIX in their classes until the operating system had its source code closed.

The copyright barriers imposed by Bell Labs in 70's pushed people to create important and amazing UNIX-like versions. Seminal books were written and projects were created since then. Nowadays, we have UNIX-like systems running not only in servers but also in your beloved smartphones, routers, IOT devices, etc. There is no doubt that the UNIX philosophy has been a huge success.

Even being a proprietary operating system, UNIX shares an important standard called Single Unix Specification. This document makes possible the interaction of different UNIX-like families, including the interchanging of programs and programming libraries.

The Single Unix Specification is basically composed of three documents: ANSI C, XPG4, and POSIX.

As suggested by the name, the ANSI C standard is related to the main UNIX programming language. Due to it, the ANSI C must be followed as much as possible when writing programs that must run in several UNIX implementations. In this way, portability will be easier. Even with non UNIX-like systems, usually the operating system has a minimal C library normally based on the standard C library. This gives us an important tip: code considering ANSI C from the beginning and you will never be sorry.

XPG4 is the standard related to the X server and graphical UNIX parts.

POSIX is perhaps the most important standard of the three. It defines the standard signals sent and received by the processes and the standard system calls. Any good UNIX-like system must be POSIX compliant.

Stacks and Queues are the most common data structures present in computing.

Lists can be understood as a generalization of the ideas introduced by queues and stacks.

Programmatically, a tree can be understood as a multi-level list.

FreeBSD and NetBSD features standard utilities for list typed data implementation.

Note

You can learn more about Device Driver Development by joining our online course on bsdmag.org

Meet the Author

Rafael Santiago de Souza Netto is a Computer Scientist from Brazil. He has been working as software developer since 2000. He usually contributes writing software for Computer Science research groups from Brazil. He has about 19 years of experience in C programming. His main areas of interest are Programming, Computer Networks, Operating Systems, UNIX culture, Compilers, Cryptography, Information Security and Social Coding. In his spare time he likes to continue writing code but also articles (talking about code) for BSD Magazine, 2600 among other publications.

Monitoring OpenBSD using CollectD, InfluxDB, and Grafana

In a “get pretty graphs” mood, I’m looking at what can be done regarding OpenBSD monitoring using the CollectD collector and Grafana dashboard renderer. OpenBSD 6.2-current provides InfluxDB and Grafana packages. A great stack for pretty reportings.

Host the data

System metrics will be stored in InfluxDB because it can be used as a Grafana source. The installation and configuration is straightforward. The key thing is to enable the collectd protocol.

```
# pkg_add influxdb
# vi /etc/influxdb/influxdb.conf
(...)
[[collectd]]
  enabled = true
  bind-address = ":25826"
  database = "collectd"
  retention-policy = ""
  typesdb = "/usr/local/share/collectd"

# rcctl enable influxdb
# rcctl start influxdb
```

Note that this service works using UDP only. Unless I missed something at the time of writing, there is no TCP nor TLS options available.

```
# netstat -na | grep 25826
udp 0 0 *.25826 *.*
```

Collect the data

I mostly use the CollectD as a metrics collector because it knows about OpenBSD, and can send its data remotely. In this case, to InfluxDB, enable any required plugins. Don't forget the network one so that data can be sent to InfluxDB.

```
# pkg_add collectd
# vi /etc/collectd.conf
(...)
<Plugin network>
  <Server "127.0.0.1" "25826">
  </Server>
  ReportStats true
</Plugin>

# rcctl enable collectd
# rcctl start collectd
```

Render the data

New in OpenBSD 6.2-current : Grafana is available as a binary package. This will enable pretty graphing using my prefered OS.

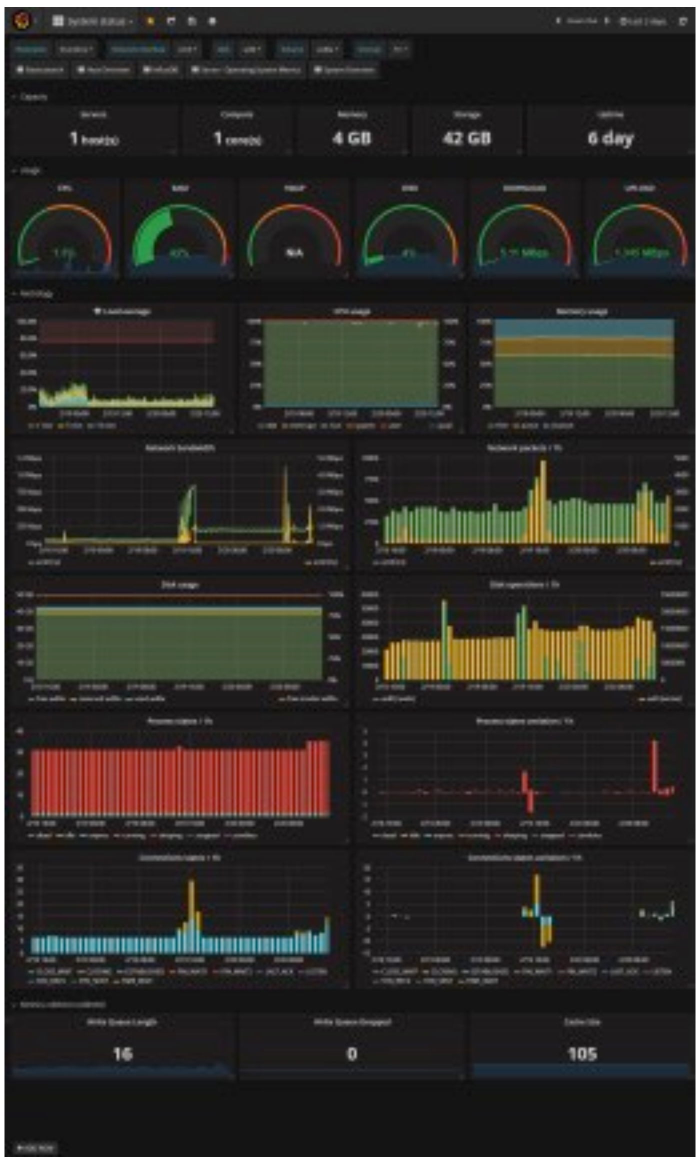
```
# pkg_add grafana
# vi /etc/grafana/config.ini

# rcctl enable grafana
# rcctl start grafana
```

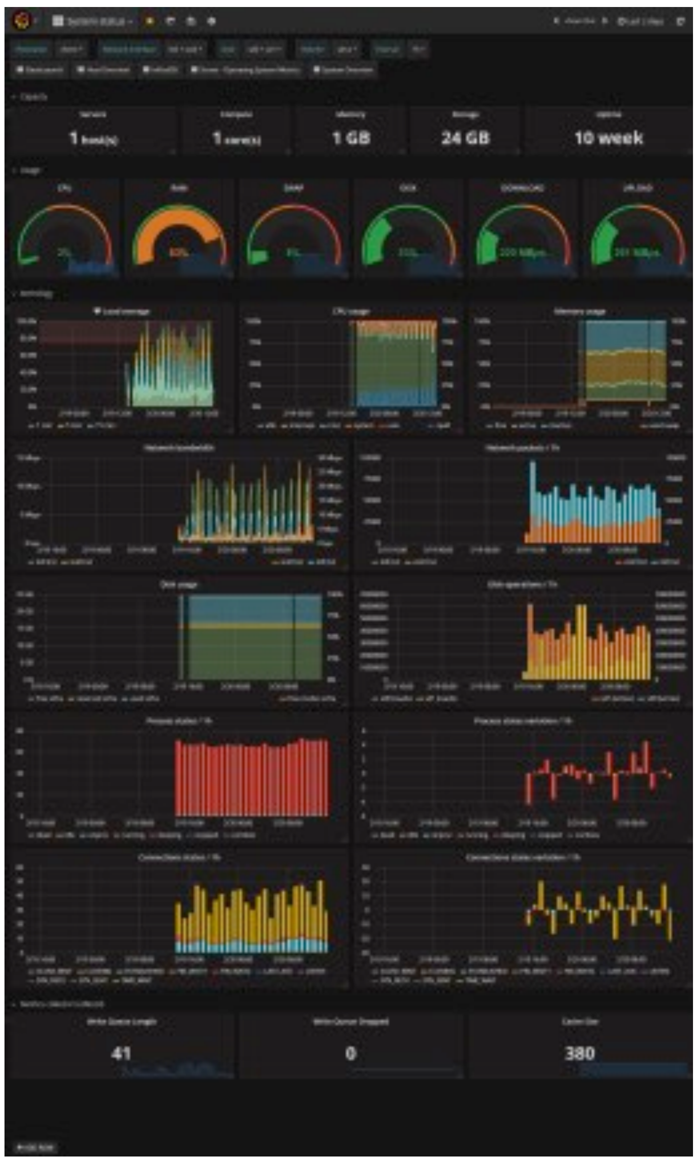
Browse to http://localhost:3000/ and log in using the default credentials (admin: admin). Those can be changed this way http://docs.grafana.org/installation/configuration/#security and from the GUI.

In Grafana, add the InfluxDB source using the collectd database.

There are example dashboards available on Grafana's website. Namely #554, #555 and #755. They will nearly work out-of-the-box and can be used as a base to create yours. They seem to be Linux-centric but here's how they look, once slightly modified for OpenBSD.

I’ve created one from scratch to render default collecting data from my OpenBSD servers. It looks like this :





Should you want to use it, I have made it available online here.

That's All Folks!

MEET JOEL CARNAT

Please tell us about about yourself?

I am a 42 years old techie, and a far as I can remember, there's always been a computer at home. I started using a Thomson MO5 when I was about 10. Then, we had a Macintosh 128K and a SMT Goupil G5. Thereafter, we had various 80386, 80486DX2, and Pentium machines running on either DOS or Windows OS. I don't remember precisely but in the early 1990's, my father brought me a book about UNIX, which shipped CDs with Slackware Linux on it. Then he brought a magazine with a CD of FreeBSD. And I was attracted by the shell. I have worked as a System Engineer since 1998, sometimes as an employee, sometimes as an IT consultant. Since 2008, I evolved a bit and served as an IT Architect for various clients. As of 2015, a friend and I started our own company. We're helping our clients to make the most of their IT systems.

My day-to-day work is to deal with Linux and Windows systems. During my time off, I practice Karate and Callisthenics, or play with some OpenBSD instances to host my personal IT services and explore things.

How you first got involved with programming? What was your path?

I started programming using Logo and Basic on the Thomson MO5 my father brought home. Then in school, I learned Turbo Pascal, and finally, in University, I learned C and JAVA. As a personal interest, I learned shell programming on Bash and TCSH.

Reading your blog, we can see that you have a wide field of expertise. Please tell us which is your favourite area?

I'm not sure if I have a favorite area. I'm more of an Ops than a Dev. And I know much about Systems than Networks. But I can do storage, virtualization, email, network services, web stuff, etc. One of my strengths is being able to deal with (nearly) any technology. I like to say, « if there's a shell, there's a way. »

It seems the OpenBSD is your favorite OS? Why? What features are the best and what you like the most?

The first reason I opted for OpenBSD, believe it or not, was because of its Puffy mascot. I liked it more than the Penguin. Thus, I learned the OS. And I was fascinated by how it is built by developers. It's simple, efficient, and clean. What is supposed to work, « just works ». There was a period when I mostly used NetBSD because of its documentation. However, I stopped using it when it started having some glitches that OpenBSD didn't have. Another reason why I like OpenBSD is its six months release cadence. It's easy to get prepared for OpenBSD upgrades. And you're not digging for things that changed on a chaotic cadence. I like the fact that the base system and the ports are separated, and the fact that most of the software can be run and managed from binary packages. No more « wait 6H for dependencies to compile ». Lastly, the feature I like most is syspatch(8). In my view, this makes the OS Production ready for the enterprise. AFAIK doesn't deal with port upgrades yet. But M:Tier's openup does the job at the moment.

What is your the most interesting programming issue you encountered, and why was it so amazing?

As I said, I'm not a developer. So I have never encountered any real programming issue. But what's impressive is the ability of the OpenBSD Dev community to manage all this software that was not designed to be running on OpenBSD.

What tools do you use most often, and why?

There's every likelihood that OpenSSH, ksh, and Vim are among the top five. Other tools I frequently use are cat, grep, less, and awk. Those are my day-to-day friends to manage, debug, improve or correct IT services.

In the top 10 list, there would probably also be Word, Excel, and Powerpoint because I have to deal with users that don't read shell. Those are honestly great tools to show-off and explain things toend-users.

What was the most difficult and challenging implementation you've done so far? Could you give us some details?

The most challenging thing I've ever done was building a whole IT system. From racking the servers in the datacenter to configuring a complete Active Directory + Exchange + SharePoint environment, while still having to setup and manage the EMC SAN and the VMware vSphere infrastructure. We were five guys working hard to set up that whole thing for 10 000 users. At that time, I only knew about Linux and OpenBSD running on independent servers. Therefore, I had to learn those new technical infrastructure layers, and understand how the Microsoft Services worked compared to the Open-Source Software I knew. In the end, all went well. It is a great memory.

Can you tell us about your favourite features in the new releases of your favourite OS?

Not really. I don't have any missing feature on my servers. They run in the Cloud, they run in the virtualisation system I own. And they work well. Since 6.3, Grafana and the ELK stack are available. Infact, they are more stable than the ones I run on some Ubuntu systems. Come to think of it, it's maybe the first time I don't expect more from OpenBSD.

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

I would like to finish my Grafana dashboards for every service I run on my OpenBSD servers. I would also like to find time to switch my WordPress instance from Apache to httpd(8). Further, I need to learn about the Amazon Web Services and it's way to implement Infrastructure as Code (IaC). That's probably my 2018 main goal.

What's the best advice you can give to the BSD magazine readers?

Maybe to just do it with *BSD whenever it's possible. Even in big companies that paid for Microsoft or RHEL support, I was able to set up a few OpenBSD boxes: a PF cluster to protect the network, a bunch of OpenBSD/OpenSMTPD servers to relay internal emails, and an OAMP farm to publish web applications. *BSD is not just for old nerds, hobbyists or SME. It can be used to provide high-quality IT services. Hence, it's important that the World should know about it.

That's part of the reasons I write articles on my blog. To prove that you can do valuable things with OpenBSD.

Thank you

Expert Speak by E.G. Nadhan

From Unconscious Bias to Unbiased Consciousness

A member of the audience attending a panel session on **Unconscious Bias** accidentally referred to the topic as **Unbiased Consciousness**. Perhaps, it was no accident and was a sublime message instead about the world to come – a world where we are consciously unbiased rather than being unconsciously biased. However, this utopian world can become real only if proactive actions are taken to combat such mindsets that may not be in our control.

What's the most challenging part of unconscious bias? It is *unconscious.* You don't even know that you are doing it while you are doing it. Yet the outcomes of your actions will speak for themselves – by which time, it might be too late.

I recently attended this panel session on **Unconscious bias and how to make your voice heard, organized by SpringCM**. I wanted to listen to and learn from the panelists on their first-hand experiences being on the receiving end of unconscious bias. During the session, I began thinking about the need for action on all fronts to combat this phenomenon.

The panel was moderated by Heather Christman, the senior director, strategy and development for PeopleFoundry, while four distinguished panelists shared their insights: Manika M. Turnbull, Ph.D., VP & Chief Diversity Officer at HCSC; Terri Brax, CEO at Women Tech Founders; Michelle Joseph, CEO & founder at PeopleFoundry; and Andee Harris, CEO at Highground.

Insights on unconscious bias

Here are some realities about bias that surfaced through the various experiences of the panelists:

- Bias exists because people exist. It is pervasive.

- Bias is activated without the individual’s control, possibly leading to snap judgments and blind spots.
- Bias grows over the years in the world around you.
- Bias is fueled in the comfort zone of working with people like yourself.
- Bias is expedient – you’re just getting the work done.
- Bias surfaces in unexpected places, such as the words used in job descriptions, and holidays that are celebrated within the enterprise.
- Bias comes across when the gender of the working parent triggers questions about parental responsibilities.

Four ways to fight unconscious bias

These insights led me to wonder what we can do to consciously combat this unconscious bias. Here are some of my thoughts on how we as leaders can fight it:

1. Groom. Human bias is based upon casual observations. We form opinions based on what we see in the world around us resulting in our brains training themselves on repeating phenomena. *That is the way I have seen it – and therefore, that is the way it ought to be.* Today’s workforce needs to have balance, for example, including people of different genders, ethnicities, and physical challenges. So does the workforce of tomorrow.

Today’s schoolchildren are tomorrow’s torchbearers and thought leaders. A healthy mix of children from upcoming generations must be trained and motivated to engage in STEM projects.

Combat Force One: Grow the diversity in the future workforce.

2. Collaborate. While enterprises can take action within their firewalls, unconscious bias is human. There are no corporate or regional boundaries for unconscious bias. As one panelist asserted, it is pervasive across the extended enterprise. Therefore, it is vital for enterprises to join forces and take action. This panel session is a fine example of such collaboration – but collaboration needs to be extended to jointly take action across the corporate and the academic worlds.

Combat Force Two: Corporations can collaborate with academia to change the DNA of the workforce.

3. Cross-pollinate. Diverse teams must be staffed with people of different mindsets – not just a segment of the community. Project teams benefit from input from a wide variety of people. (We have heard some CIOs call this bringing “texture” to a problem-solving team. The texture - and problem-solving power – of the group increases with the diversity of voices and ideas.) For example, the fine panelists for this session (and the moderator) happened to be women who shared great insights, triggering a thought-provoking conversation. Cognitive diversity is not about who you are but how you think.

Combat Force Three: Rethink how you construct teams, keeping unconscious bias in mind.

4. Measure. Subjective conclusions are extremely difficult to measure. *How much do I like a person? Or not?* Quantifiable performance outcomes matter. How is the overall performance of the enterprise affected by gender diversity? Evidence (like this McKinsey research on diversity and corporate profits) shows the positive impact diversity has on the overall financial performance of an organization. (See also what MIT Sloan Professor Thomas Malone's research says about high-performing teams.)

However, it is important that outcomes are measured, tracked and communicated at your enterprise – to spread the information about the resulting benefits.

Combat Force Four: Quantify the performance of the enterprise.

I am sure there are enterprises who are already taking one or more of these steps. Do other solutions come to mind? Please let me know.

Meet the Author

E.G. Nadhan is the Chief Technology Strategist for the Central Region at Red Hat. He provides thought leadership on various concepts including Cloud, Big Data, Analytics and the Internet of Things (IoT) through multiple channels including industry conferences, Executive Roundtables as well as customer specific Executive Briefing sessions. With 25+ years of experience in the IT industry selling, delivering and managing enterprise solutions for global corporations, he works with the executive leadership of enterprises to innovatively drive Digital Transformation with a healthy blend of emerging solutions and a DevOps mindset. Follow Nadhan on Twitter and LinkedIn.

Column

With Facebook attempting to slam the privacy stable door well after the horse has bolted, the corporate giant has suspended over 200 applications which snarfed large amounts of profile data. What does the future hold for this global platform?

Rob Somerville

I have a certain degree of sympathy for Mark Zuckerberg after being hauled before Congress in light of the Cambridge Analytica fiasco. Inevitably, any cutting-edge technology will eventually feel the hot breath of the establishment breathing down on it, be it via indirect legislation or as in the case of Mark Zuckerberg, in personal appearance before “the powers that be” to give account.

If I was Mark Zuckerberg, I’d be worried. There is a Shakespearean idiom that fits this scenario perfectly - “Hoisted with his own petard” - which means in plain English, to be blown up by your own bomb. There can be few Facebook users that are ignorant of the whole sorry story, no doubt spread and amplified and echoed via social media. Irrespective of the moral rights or wrongs here, you can be assured the accusations surrounding Facebook will at the very least, lead to further government regulation, and at worse, to the atrophying of the platform to such a degree that it will be sold off or split up, not unlike the fate of IBM and the large US telecommunications companies. Or indeed, both. Irritated governments in the media spotlight have a habit of using blunt instruments, and as we well know, politicians and lawyers by nature don’t have a good grasp of the fundamental issues surrounding IT.

I don’t think Facebook will suffer enormous censure out of this investigation, they are, after all, an American company. The closest I can remember in history of this battle of giants was when lawyer Ralph Nader, appeared in Congress to testify about automotive safety. His advocacy lead to the adoption of the 1966 National Traffic and Motor Vehicle Safety Act, which forced car manufacturers destined for the US market to equip vehicles with padded instrument panels, seat belts, and reversing lights. This forced both the US and foreign manufacturers to take safety seriously, certainly as far as the US market was concerned.

Personally, between fake news, the continuing increase in scandals surrounding child pornography, and the focus on “hate speech” on social media platforms, I believe we are very close, or if not actually, reaching a tipping point. Big government is getting more interested in big data. We are on treacherous territory here, as the Internet is considered the bastion of free speech and expression. The danger here is that Mark Zuckerberg has invited a close forensic examination of not only what Facebook is, but also what it does, by an establishment that by design can only output a bland consensus. It might end up with some common-sense legislation like the 1966 Act, which would be a good thing. It might not, however. Facebook undoubtedly needs its wings clipped, and Mark Zuckerberg’s appearance in front of Congress will have been a sobering moment for the disrupter CEO. How much, in reality, will this carry through to Internet culture is a different matter entirely.

It is all very well when Facebook shouts from the rooftops the benefits they bring as a social media provider. There can be no argument about the blessings they have brought to families, individuals, and communities. They have also been responsible for a phenomenal amount of heartache, from the spouse who has found out about their cheating partner, to the interview candidate who missed their dream job due to a prior HR search. With power, comes responsibility. Playing about with the developers API a few years ago confirmed my suspicions. Anyone with some decent kit could harvest this data wholesale, and use it for nefarious purposes. Having seen UK supermarkets “data power” gathered from point of sale datasets and loyalty cards prior to 2000, I shudder to think where Facebook rests in 2018. Facebook needs, desperately, to get off the “we are a just a provider” fence. Hopefully, Facebook might actually be facing some social responsibility and accountability for a change.

If Facebook wants to make a name for itself, they have a lot to learn. For instance, they might just want to, occasionally, not hand data over to law enforcement or intelligence agencies, carte blanch. The UK Guardian newspaper went through the surreal act of destroying hard drives with drills and angle grinders in the presence of the security services, to protect Edward Snowden. That is data privacy in action. Of course, Facebook is not a publisher, just a medium, or so they say. Where do you draw the line between community service and private privacy if you are not a journalist? On the balance sheet. There will be a bunch of individuals within the organization who understand the importance of the trends reflected in the data, and that will be leveraged to maximize the profit. Which is why I don’t agree with tax that is applied to carrier bags. I agree that the planet is a precious resource, and we need to cut down on plastic. Most carrier bags these days carry a brand logo. Having to pay to advertise for a corporate adds insult to injury. The same principle applies to Facebook, it might be free, but there is a price to pay.

Facebook *per se* is a corporate marketing department’s wet dream. Be friendly, build a community, and gather intelligence. Sorry, information. Sell, or leverage this data. Job done. Money made. I have a teenage daughter who has pleaded with me not to look at her twitter feed. My wife, after discovering some inappropriate comments on said daughter’s Facebook page, called her to account, and I was in full agreement. After reading what was said, I really wish I had listened to my gut feeling and pulled out the plug on my daughter’s social media access at an earlier point. The fact my daughter is ashamed to share with her father what she types online, says something about the type of disconnect that Facebook actively cultivates. I am too much of a gentleman to throw my little weight around, and I am waiting for reality to kick in.

Facebook is now getting into the dating market. The Wall Street index had initially placed their bets on FB. The Internet dating agencies stock has fallen. So, what's my advice to isolated IT professionals? Make real life friends, develop a real social circle, and your friends will sort it out. Until you hear or see someone eat, you don't get it. You will never get that on Facebook, and if you could, you can still act or lie. Currently, said daughter has met her first boyfriend online, and as far as I can tell, he isn't a deadbeat. Where this relationship goes, God only knows. The desire to crack them both on the head with the realities of data privacy, potential psychological damage, and realities of life is a constant temptation, but they are both from the "plugged in" generation. A few hours without Internet access and like heroin addicts going cold turkey, they will be invariably climbing the walls. Books, music, and face-to-face interaction are an anathema to them both.

In reality, Facebook has diversified enough that even if their core platform becomes a pariah, as a technology company, it has an exit strategy. The question on everyone's lips is a simple one – Facebook may have been a global social media electro-magnet, attracting the iron filings of our lives, but where will that data turn up when the power is turned off?

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.

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