My BSD sucks less than yours

Baptiste Daroussin
bapt@FreeBSD.org
The FreeBSD project

Antoine Jacoutot
ajacoutot@OpenBSD.org
The OpenBSD project

Abstract

This paper will look at some of the differences between the FreeBSD and OpenBSD operating systems. It is not intended to be solely technical but will also show the different "visions" and design decisions that rule the way things are implemented. It is expected to be a subjective view from two BSD developers and does not pretend to represent these projects in any way.

We don't want it to be a troll talk but rather a casual and friendly exchange while nicely making fun of each other like we would do over a drink. Of course, we shall try and hit where it hurts when that makes sense. Obviously, we both have our personal subjective preferences and we will explain why. Showing some of the weaknesses may encourage people to contribute in some areas.

Most of the topics discussed here could warrant their own paper and talk and as such some may not get the deep analysis they deserve.

This is a totally biased talk from two different perspectives.

Speakers presentation.

The History behind the story

aja

Before we begin, let us tell you a little story about how we ended up before you today. Once upon a time on a sunny afternoon somewhere in the countryside, Bapt and I were having some herbal tea and sponge cake...

bapt

We often meet together to share our thoughts and vision about what's going to be the next big thing in IT or have long and philosophical discussions about technology in general. That's how we predicted the Cloud before that word was even invented.

aja

Indeed, I still remember when we told people about Docker before Solomon was even born or how macOS would stop being useful years before it stopped being useful!

bapt

Absolutely! Anyway, it was around 5pm on that beautiful afternoon and… OK who are we kidding here, I think it was more like 2am.

aja

Yeah… we were groveling in some random bar having our 10th glass of Californian Zinfandel when...

bapt

Burgundy, it was Burgundy. Be careful this is France, you are not allowed to say such things here!

aja

Oh right sorry! So I started complaining that the Burgundy didn't taste anything...

bapt

No no no, I told you can't say things like this. So yes, no taste, maybe it was Californian Zinfandel after all.

aja

So, as I was saying, the wine did not have any flavor anymore. And that's when it hit me… Flavor… wait a minute… OMG this is exactly like FreeBSD. You guys don't have FLAVORed packages!

Trolls must now leave the audience.

ACT I

"Let's get ready to ruuuuuuuuummmmmmmmmmmble!"

Scene 1: Ports & packages

aja

FLAVOR is an awesome concept. I am not aware of any other package manager in UNIX land that implements the same functionality. Basically, it allows providing packages compiled with a different set of options. That's very convenient from a dependency point of view since according to the options you want, the final binary will change as it might end up linking to different libraries.

So you want to be able to control that but providing packages compiled with a good set of defaults can be next to impossible sometimes (it's hard compromise) and that's where FLAVORs kick in. For example you can install Sendmail with or without support for SASL and / or LDAP: there are different (conflicting) packages for it (e.g. "sendmail" vs "sendmail--ldap-sasl").

It's different from subpackages which are just split packages. These are typically used when you have a software that comes with a huge amount of data (like audio and graphics for games for example) which rarely change while the main binary does each update. Instead of having to update the complete set, you just update one small subpackage. It can also be used for things that ship modules or add on components that have a whole lot of different dependencies. PHP is a good example. On OpenBSD we compile it with pretty much all options, so the build requirements are big but the port only needs to be built once and since the package is split into multiple ones, you only get to install what you need (e.g. php-imap, php-mysql…).

bapt

WIth that last one, you really took the wrong example here. For php we have fine grained split the packages so if you want php with imap? Just pkg install php5-imap and done.

aja

But my point is actually that subpackages allow you to not have to deal with fine grained packages. If I understand correctly how your ports tree works you still need to build each PHP module separately one by one, while we do it all at once.

bapt

Correct, these are not really subpackages and would be nicer with them. For FLAVORs a good example could be the OpenLDAP client compiled with or without SASL :)

aja

Right. So if I need SASL support in OpenLDAP on FreeBSD, I assume I need to build it myself right? That’s fine I guess, but my fear is what will happen when I run “pkg update && pkg upgrade”?

bapt

Bad things...

So yes it is true that we do not yet have FLAVORs nor subpackages but there is some ongoing work to provides those features in the ports tree (actually in final review right now). That work is pretty hard as it breaks the design of some tools that are heavily used by our users like portmaster and portupgrade and which are barely maintained; meaning we need to find an upgrade path which will not break those. We try to always have an upgrade path for users and try to avoid hard breakage. Keep in mind that we do not release a package set, but the ports tree is a rolling release not tied to a FreeBSD release. Which reminds me, you guys do not support upgrading packages on a given release!

aja

Actually we do support upgrading packages on a given release. What we do not provide are precompiled binary packages. So it’s true that you need some kind of a build box where you can distribute packages updates from. But there is work in progress in this area and we should be able to provide updated binary packages in the near future; we just need to agree on the workflow, the team responsible for it and get the required infrastructure.

Talking about pkg_add(1), it’s important to note that a lot of operations are done as different unprivileged users (fetching, extracting etc), we don’t really want to go on the Internet as root while you guys still do not drop privileges when using libfetch in pkg(9). Care to comment about that?

bapt

That is not true actually pkg(8) uses capsicum to sandbox everywhere it is possible. And pkg(8) also uses unprivileged user to fetch packages for examples and everywhere it is possible. While sandboxing is supported for a long time, unprivileged users mechanism were added recently before OpenBSD did it … by a few days :)

aja

Yeah but the libfetch commit was reverted...

Back to our topic, we also try very hard to provide proper upgrade paths for our users, but the difference is that we don’t have to deal with legacy / unmaintained wrappers against the pkg tools (portmaster, portupgrade, portinstall, portguru^wwhatever...) . Our pkg tools are designed to provide everything one needs for package management without the need for an external helper. Don’t forget we have a long tradition of providing proper binary packages over pushing people to compile their own.

bapt

This is also true for FreeBSD now, most of the tools you are speaking about are mostly to use the ports tree directly as a live system rather than using binary packages (which is a usage not supported by OpenBSD right?). Now freebsd first class citizen is binary packages

aja

It's perfectly supported by OpenBSD. We just don't encourage it it because for 99.99% of the cases, it is not needed.

bapt

But one thing about the FreeBSD ports tree is that it works on all supported versions of FreeBSD. Ah well... true you only support one version of OpenBSD and only for 6 month.

"DING!"

Scene 2: Release model & engineering

aja

You are correct, for packages we only support the current release which at this time is is 6.1; the operating system itself is supported for two releases.
And aside from current/snapshots, our ports tree is indeed tagged to a specific release. That's a design decision. We do not want our users to be forced to upgrade to a new major version of a software on a given release and we do not want to maintain 4 different versions of the same software to prevent that. Which is why our ports tree does not follow a rolling release model (except for -current obviously). Considering the fact that we have regular releases every 6 months, our packages don't have time to get too old anyway.
Our support policy is pretty easy:

- Base: n and n -1
- Packages: n (ongoing discussion about supporting the previous release as well)

It's true that this means you must upgrade at least once a year if you want to run a supported release. I don't think that's a bad thing: I'd rather upgrade often in small steps than once every 5 years having to handle multiple invasive changes at once.
I do understand the need for LTS releases, I just don't like it when I am the one responsible for upgrading them...
That said while it can be good to have a some long term support WTF is your policy?
Extended support versus normal support?
How come FreeBSD 10.1 was still supported while 10.2 wasn't?
This is all very confusing, no support can be good if you don't understand how it works.

bapt

Common that is easy, let's read it:
<u>Normal:</u>

Releases which are published from a -STABLE branch were supported by the Security Officer for a minimum of 12 months after the release, and for sufficient additional time (if needed) to ensure that there is a newer release for at least 3 months before the older Normal release expires.
Extended:
Selected releases (normally every second release plus the last release from each -STABLE branch) were supported by the Security Officer for a minimum of 24 months after the release, and for sufficient additional time (if needed) to ensure that there is a newer Extended release for at least 3 months before the older Extended release expires.
Easy no?
Nah… I agree this is a mess, further more it was causing lots of issue when maintaining the ports as we had to wait for at the very least 2 years before being able to use modern tools available in newer base. Also issuing a new release for important fixes that did not fit en errata was having a huge impact: extending the support for the branch for at least 2 years.
Actually I should have used a past tense, because this has changed recently with FreeBSD 11.0 the model has changed: each major version of a table branch is explicitly supported for 5 years, while each individual point release is only supported for three months after the next point release.is out Given we ensure binary compatibility on a given major branch, this allows users to benefit stability for 5 years, and at the same time the benefit new features.

aja

It looks like FreeBSD support model keeps newer features out of the hands of users because it can be years before they actually see them in a release. It reduces real-world testing and benefits no one. In my opinion, OpenBSD model is easy to understand and keeps the code stable but still pretty fresh. Whatever is in current at a certain period in time will end up in the next release (which would take at the very worst, 6 months).

bapt

You seems to have misunderstood the new release model, new feature not breaking binary compatibility are merged into the stable branch pretty quickly and we will be able to issue new release on those branch whenever we want to have releases supporting those new features.

aja

Well you did say that you “had to wait for at the very least 2 years before being able to use modern tools available in newer base for ports”. We do things differently, we are not afraid of breaking binary compatibility in -current and for our upcoming release (no merge / backports necessary nor wanted). But as you put it, your process is a mess so yes, I may have misunderstood.

“DING!”

Scene 3: Binary upgrades

bapt

Given we do support binary upgrades it easy to upgrade from a minor to another minor (by the way binary upgrades works also nicely across major releases) but hey, here I must be losing you as you do not support binary upgrades neither for security fixes, neither for upgrade across releases, seriously how tedious it should be to have OpenBSD boxes in productions?

aja

While I see where you're going, the way you put it is not entirely true. Binary upgrades between releases are perfectly supported. What is not supported are in-place upgrades of the base system from one release to another. But is that really an issue? Supporting in-place major upgrades would not prevent us from having to reboot anyway.
The upgrade process of OpenBSD is one of the easiest and fastest I've ever encounter. Just reboot on the new release bsd.rd (ramdisk kernel) and it will do the rest for you in like 5 minutes. Also, thanks to our auto-install / auto-upgrade functionality, you can give it the path to an "answer file" (via dhcp) and you won't even need manual interaction. So you can rely wipe and reinstall or upgrade an unlimited number of machines in a snap. And it can also upgrade your packages to match the new release! Add this to the fact that sysmerge(8) (our mergemaster equivalent) is automatically run early at boot time after an upgrade, you are really left with almost nothing to do…

bapt

Maybe but that only works from release to release (or -current), not when you want to keep your currently running release up-to-date with security patches.

aja

That was true until a few months ago: updates within a particular release were only provided in the form of CVS patches. But this all changed with the 6.1 release and the new syspatch(8) utility which is a base system update tool that will fetch, verify and install binary updates (i.e. tarballs containing the updated binaries / libraries…). Its design tries to match our installer: fast, simple, stupid and safe (in no particular order).

bapt

On the FreeBSD side we do support in place binary upgrade via freebsd-update. The basic design behind freebsd-update is to create binary diffs and binary patch your system.
It is very convenient to receive security fixes.
freebsd-update also allows to upgrade FreeBSD across releases "in place" the procedure is then the following:

- First install the new kernel
- Reboot on it

- Install the userland
- Merge configuration files
- Remove the files not in the release anymore

It works very nicely but fetching all the binary patches when upgrading can be very very long. Another issue is that producing the patches is really not trivial. That is why we are looking forward to be able to provide a new mechanism based on pkg(8).

To be honest there is also a drawback about in place upgrades we need to maintain strong backward compatibility on some kernel interfaces vs the related userland tools. In particular ZFS is an interesting place. Illumos consider the version in kernel goes with a given version of the userland tools. So if they modify a kernel interface they modify the userland related tools to use that new interface and this is done. On FreeBSD we have to add a compatibility shim in the kernel because otherwise rebooting on the new kernel one would end up with userland tools that does not work with the kernel until the userland itself as also been upgraded.

aja

In-place upgrades can be interesting when the upgrade process itself takes a while. In theory you would prevent long down times. But in reality, production should be resilient and services should be redundant so it’s generally not such of a problem. Also the release upgrade process on OpenBSD is literally the fastest one I’ve ever used.

If in-place upgrades meant updating and reloading the entire userland and kernel without rebooting, now that would have my vote!

bapt

That said, about upgrade path, an interesting policy we have on FreeBSD is POLA (Principle Of Least Astonishment) this is a strong policy on FreeBSD saying we should try very very hard to make upgrades as seamless as possible and avoid breaking end user boxes as much as possible.

aja

In general we are not afraid of breaking backward compatibility when we think it makes sense and help push ideas forward. That said I can understand the need for a POLA but not when it goes against basic security. For example, just to satisfy third party clients you guys kept DSA encryption, AES-CBC cipher and SSH1 support in your OpenSSH a long time after upstream (aka OpenBSD) dropped it. That is the kind of compromise we are not willing to make.

bapt

That is not entirely true. POLA does not prevent from making such modifications, but they should be driven as in the users should be warned long enough before. This is in particular true on a given branch, big changes usually happen in new major branches. On newer branch they were just removed.

aja

Well, according to FreeBSD SVN revision 303716 (Wed Aug 3 16:08:21 2016 UTC):

“Remove DSA from default cipher list and disable SSH1. Upstream did this a long time ago, but we kept DSA and SSH1 in FreeBSD for reasons which boil down to POLA. Now is a good time to catch up.”

“DING!”

Scene 4: Package building & delivery

aja

Speaking of upgrades delivery, it would be interesting to know how you guys build and ship packages. On OpenBSD we use a tool called dpb(1), the Distributed Package Builder. It’s a perl daemon that orchestrates a multi-node setup for building packages. It allows us to provide -current binary packages daily (in reality we only ship new packages every ~3 days unless there is a good reason).
As most OpenBSD daemons, it comes with privilege separation: a different unprivileged user is used for fetching, building and doing specific builder tasks. While pkg_add(1) still runs as root, dpb(1) allows building each package in its own chroot(2) using the proot(1) tool; the end goal being to actually build each package under its own sandbox on the fly. Also it’s worth noting that on our official build setup, the build user is prevented from reaching the network; the only process that can go online is the one that fetches the distfiles for building from ports (and that process runs as a different user).

bapt

Historically, we were using a tool which a bit like dpb was distributing the builds across multiple machines. This has been changed years ago when we switched to poudriere which is designed to work on a single box distributing the builds across multiple CPU cores. Doing that we have benefit faster builds and way simpler code. The packages are signed on a box without internet access, the main difference is probably that we do not sign every packages but the repository metadata which contains hash of all the packages.

aja

Same here, all of our packages are cryptographically signed using our signify(1) tool on a dedicated machine that has no access to the internet. The difference is indeed that we do not want to rely on metadata so the packages themselves are signed; it also makes it easier for sharing packages.

bapt

Since we switched to poudriere we are able to provide updates of binary packages almost everyday for the stable and head branches of the ports tree.

While working on that we have leverage all FreeBSD features:

- Jails: all build are done inside a jail without network access

- ZFS: to fast populate the jails and ensure we are always building in a clean room
- TMPFS: for fast I/O
- And SMP: yes we can use more than 24 cores :)

"DING!"

Scene 5: SMP & scheduling

bapt

Speaking of using multiple cores, you are still using the Giant lock almost everywhere? That should be a big problem for you to scale isn't it?

aja

For a regular "Desktop" usage, our "Big lock" SMP implementation model (KERNEL_LOCK()) is good enough. The reason is that most of the time on a workstation you only have a handful amount of cores (between 2 and 8) and only one socket. Our actual scheduler isn't bad, it's just that it is a bit old and was written for real SMP machines. Meaning it does not consider the cache distance between cores and that's the main reason why machines with several sockets often have lower performance than the ones with only one socket (less ping-pong involved).
So we can also use 24 cores, but in the context of bulk builds, we have much better performance concatenating the work of 6*4 cores machines for the time being.

bapt

How such a change can be done in the OpenBSD model, I mean on FreeBSD it took a lot of time to get most of the architectural work landing it took us several years including releases that got very bad reputation :) FreeBSD 5. I'm not saying the road we took is the one true way, but I can't see how this can be achieved in an incremental way.

aja

A few things can be incremental because they are not directly related to the scheduler itself. Several scheduling issues on OpenBSD came in fact from spin locking in librthread (our POSIX 1003.1c thread implementation) but that has drastically improved since we implemented futex(2) and make our pthread library use it. And contrary to the myth, OpenBSD scales very nicely on big userland workloads. Now, is raw performance on par with FreeBSD? No, but our priorities are different and we are OK with losing a bit in this area if it gains us more security and/or simplicity. Actually I'd go further and even say that we'd rather crash / panic if we detect an unsafe behavior.

bapt

To be honest SMP is something that is always evolving, right now our next challenges are better NUMA support, and improved the overall locking systems using lock less mechanism where possible via for example Concurrency Kit. What is the current status on OpenBSD?

aja

Well our entire SCSI stack and kernel profiling are already fully SMP for example and recently there has been some huge progress on making our network stack more SMP friendly. The goal of the NET_LOCK() being able to use multiple threads to forward traffic.
I don't think you will argue that modern scheduling is hard. Most operating systems had to do it then re-do it multiple times. We are learning from others' experience and as usual with OpenBSD we are trying to implement something simple that will match the project goals. As you said, it wasn't an easy road for you. That explains why sometimes we may seem behind on certain aspects, we take our time to do it well according to our standards, or we don't do it.

"DING!"

Scene 6: The base system, part I

bapt

While you are here, you can maybe explain something I do not understand: you seems to be keeping writing your own tools when there are already BSD licensed counterparts: httpd, smptd, vmm, etc. Aren't you suffering from the NIH syndrome?

aja

That's actually a very good question. I am glad you asked because there are IMHO objective reasons to do that. The first one would be trust: we have a coding style, practice and process that make us confident in what we develop. Another reason is also control: we know someone will not decide to change the software license one day to the next or start adding knobs for each and every crazy corner case that is out there that only two people would use. For example, looking at how many CVEs impacted ntpd or OpenSSL in the last couple of years should be a good hint as to why OpenNTPD was created and why LibreSSL was forked from OpenSSL. If I am not mistaken, a lot of your security advisories have been related to the NTP daemon included in your base system (known as the "reference implementation"); is it really having NIH syndrome wanting to write a secure implementation?

bapt

Right it makes some kind of sense for the one you talked about, as there were no "BSD quality" alternatives, but what about bhyve? It is working it can "easily" fit your security requirements about security and sandboxing. So why yet another implementation?

aja

There has been an initial porting trial a few years ago but compiling a single file ended up being a huge amount of work (let alone work); when you're confronted to such code, you need to ask yourself whether it's not worth just creating something from scratch instead of porting. The

situation may have changed since (see xhyve: the lightweight virtualization solution for OS X) but that was one of the reasons vmm(4) was created.
I would add that it also helps us providing a base system that in our definition “Just Works” without having to depend on the choice done by external projects.

bapt

Right but another example would be capsicum, you decided to bake your own sandboxing mechanism instead of porting it.

aja

There is actually a very good reason for that but since you brought up the subject of security, let’s talk about that first and compare sandbox implementations later on.

“DING!”

Scene 7: Security & sandboxing

bapt

Let’s start with our malloc implementation, while it is not a direct security related tool it helps a lot driving your development in a safe way. One developer can fully tune it to find out memory leaks via prof_leak (very similar to what you can get with gperftool, heap profiling, and way more things, you can activate all this via malloc.conf and/or MALLOC_CONF environment variable.

aja

Our memory allocator has a lot of countermeasures as well. Some are not enabled by default because they are very costly in terms of performance but developers often run with them enabled because it allows catching several issues beforehand.
The "S" option in malloc.conf(5) is perfect for this: "Enable all options suitable for security auditing."
"Guard pages" can be enabled, these provide overruns detection by creating an unreadable, unwritable page "fence" after malloc(3)ed chunks (accessing that region will trigger a segmentation fault).

bapt

So it sounds like both malloc are quite in the same shape regarding debugging and memory related potential security issues. But in a different way.

aja

Most daemons run with privilege separation: that is most of the code is run chrooted and as a non-privileged user. OpenSSH led the way on this topic. They also use privilege revocation for dropping privileges as soon as possible.

A good and simple example to look at is ntpd(8): it is a standard OpenBSD daemon so it is written with principle of least privilege in mind and so not only does”privsep”and “privdrop” but in addition to this: it has a completely privilege separated tls speaker for the "constraint" feature with no memory sharing nor fd passing and with a limited amount of allowed system calls thanks to pledge(2). The constraint feature makes ntpd(8) query the time from trusted web servers over HTTPS reducing the risk of a man-in-the-middle attack by comparing that time with the one that the remote NTP server gave us.

bapt

For ntp you are right but at the price of having a half baked implementation, one that cannot authenticate the peer it is receiving the time from (using the standardized mechanism), and miss half of the features ntp is supposed to provide (yes I agree most people won’t care). That said I would love that we get rid of ntpd.

aja

You just described exactly OpenNTPD’s strength; it actually does not need to authenticate thanks to “constraints” and it does not need to implement the kitchen sink because no one cares and if one does, then it can install the reference implementation. It’s not always about math and algorithm, sometimes “good enough” really is good enough.

On a related topic, OpenBSD is well known for it numerous exploit mitigation techniques. It's important to note that all of these have been enabled by default for years and are very hard or even impossible to disable. In no particular order, we have (non-exhaustive list):

- Address space randomization (ASLR): randomly arranges the address space to prevent buffer overflow attacks
- W^X (memory page can be either writable XOR executable)
    - in the kernel since years
    - now in userland (“wxallowed” mount option on /usr/local for now)
- ProPolice (Stack-smashing protection using bounds checking and canary)
- Position Independent Executables (PIE) (also for static binaries): the executable is loaded at a random address instead of a fixed one address which stops return-to-libc type attacks against functions in the main program (another space randomization technique)
- Strong random generator
- KARL (Kernel Address Randomized Link), which generates a unique kernel on each boot with random internal structures, preventing leaking internal kernel functions, pointers, or objects; similarly, we relink both libc and libcrypto at boot time to randomize the order of symbols
- trapsleds to limit availability of attackers to exploit ROP (return-oriented programming) by jumping randomly in programs

Each of these could warrant their own talk so I won’t go into details but I want to insist on the fact that the way we approach security is that we always assume to be running in a hostile environment. That’s also what started our constant source audit (which started in 1995, and still

ongoing) which allowed us to detect unsafe patterns and / or bugs and fix them everywhere in our tree.
We also refuse to implement “dangerous” functions, like wordexp(3); libc spawning a shell to perform word expansions, what could go wrong...
I invite people to have a look at the “innovations” page on the OpenBSD web site to see a summary of some of the security mitigations techniques that have been implemented.
In FreeBSD such mitigations seems next to nonexistent, even your (uncommitted) ASLR is intentionally weak (using non-aggressive settings). Is that true?

bapt

We also have strong random generator and Propolice. While we do not have ASLR yet, I can't argue about the HardenedBSD implementation but I would be surprised if it is true given they implemented the PaX recommendation as far as I know while PaX seems to consider the OpenBSD implementation as an "incomplete" implementation which should be named ASR apparently. By the way we have an implementation of ASR currently in review we should hit the tree: https://reviews.freebsd.org/D5603.

On our side we do have the MAC framework which is a very interesting: Mandatory Access Control which allows you to restrict many things.
With MAC we can:

- Restrict entirely what a user can see in the system to nothing but its own resources (including not seeing sockets open by other users for example)
- Have a firewall-like policy on file systems
- Restrict network interfaces access that plays nicely with IDS
- Limit the scope of the process one can see compartmentalizing them into partitions
- Restrict the information flow in a running system (very handing for example to ensure things like nagios can only access what an admin planned to make it acces and nothing more.

Another very interesting security feature we do have OpenBSM: a full feature audit system which can monitor everything that has been done on a system. It can also distribute the audit log to a central server.

aja

All these technologies you mention are interesting in theory, I agree. But in real life, I almost never see them in action because they are too complex to use and next to impossible to audit. It’s not a FreeBSD specific problem by the way, Linux has similar facilities and most people disable them…
What about LibreSSL? Are you guys planning on integrating it? Because just from a regular user point of view it has been a great success:

- usually only a handful of the CVEs impacting OpenSSL impacts us
- provides new crypto algorithms like CHACHA20-POLY1305 and X25519

We are seeing more and more people and companies adopting it in their projects and agreeing to work with the “So and so from OpenBSD”.

bapt

LibreSSL is pretty well integrated into the ports tree even if not the default. For the base system, the problem is LibreSSL development model does not fit with our ABI compatibility rules. The approach in the future would be probably to drop OpenSSL in favour of either LibreSSL or any more lightweight alternatives like mbedtls into a “privatelib” aka not exposed to the ports/packages so ports / packages can have the flexibility to use whatever tls stack we need to have at the time which yes can be libressl.

Let’s go on the sandbox part, Capsicum is a very nice sandboxing mechanism we have in FreeBSD leveraging the concept of capability. What is really nice about it is once you are inside a sandbox, there is no way to exit, even child process inherits the capabilities.

It is really designed for developers to secure their own applications by strictly restricting the capability of the application to only what it needs to be able to do. We have started converting most of the base system to use it. But we are not entirely there yet. Because the capsicum design allows no compromise, it is not always easy to convert to existing software to capsicum.

aja

That’s where I wanted to go actually: it seems that capsicum is too complicated and that’s why it doesn't get much use by default.

The pledge(2) syscall is actually a very good summary of what the OpenBSD project is about: develop practical and affordable security; meaning simple and easy to implement and hence enabled by default almost everywhere in the system. 30% of the base system was pledged after only two month. Today ~85 to 90% of base and even some monster ports like Chromium… are pledged. It was designed to be easy for the programmer to add support for it in his software, otherwise no one would use it. I doubt adding capsicum to such softwares is as trivial.

Anyway, pledge(2) itself is designed to serve two goals: encourage refactoring towards safer models and reduce code exposure to the kernel: it encourages the developer to audit and understand the software initialisation and mainloop to add pledge(2) calls where needed and move code around. One can assume that anything making its way into the OpenBSD base system will end up being pledged. People often compares it to seccomp-bpf on Linux although pledge(2) is not a system calls firewall but rather a facility to explicitly allow a group of system calls in the form of promises (sort of predeclared profiles, e.g. inet, chown, dns, tty). It can also inspect arguments. A pledged process is forced into a restricted-service operating mode where abilities can be reduced but never regained and if a restricted operation is attempted, then the process is killed with an uncatchable SIGABRT.

In theory capabilities-based security may seem more advanced, and it probably is. But what good is a feature if it is too complex to be widely implemented?

bapt

Another security feature we do have for a while are jails, it allows to create a prison/container, we have it in FreeBSD for very very long, and it very simple and easy to use. Almost everything can be disabled (and is by default) or restricted inside a jail: network, file system, CPU, memory, routing tables, etc. Nowadays jails can even be nested, have they own zfs datasets, have virtual network.

aja

Well in that regard I agree that we suck. OpenBSD has no container-like technology. There has been an initial effort years ago called sysjail but it was abandoned because of an inherent design issue in systrace(1) on which it was based that would allow evading a jail by exploiting a race condition.
Anyway I think it’s time to move on to something else.

“DING!”

Scene 8: Project governance

bapt

Good idea, let’s talk about project governance and funding for example.
The FreeBSD project is a community driven project, the active developers elect every 2 years a new core team of 9 people responsible for deciding the project's overall goals and direction as well as validating new committers proposals or the projects bylaws.
The FreeBSD foundation is a 501, US based, non-profit organization dedicated to supporting and promoting the FreeBSD Project and community worldwide. Funding comes from individual and corporate donations and is used to fund and manage projects, fund conferences and developer summits, and provide travel grants to FreeBSD developers.
The Foundation purchases hardware to improve and maintain FreeBSD infrastructure and publishes FreeBSD white papers and marketing material to promote, educate, and advocate for the FreeBSD Project.
The Foundation also represents the FreeBSD Project in executing contracts, license agreements, and other legal arrangements that require a recognized legal entity

aja

To quote its website: “The OpenBSD Foundation is a Canadian not-for-profit corporation which exists to support OpenBSD and related projects such as OpenSSH, OpenBGPD, OpenNTPD, OpenSMTPD, LibreSSL, and mandoc”.
It is a fundraising entity only: it does not own any copyright over the code even for sponsored people. It’s responsible for funding the day to day needs of the project, hackathons, etc and works solely through donations (from users and companies). It has no influence over the project itself.
Speaking about the country of origin, while it may not seem like an important detail at first it actually is when it concerns the operating system. OpenBSD being based in Canada means that we are not subject to the US crypto export regulations and all. As a consequence, it was illegal to re-export OpenBSD from the USA early on and no US citizens were allowed to work on crypto in the project.

If you wonder why we ended up with our own ACPI implementation instead of Intel's, that one of the explanation as well. At the time it was unclear whether their implementation contained embargoed code or not because they triple-licensed their code and the Intel license explicitly stated that "*Licensee shall not export, either directly or indirectly, any of this software <...> to any country for which the United States government or any agency thereof requires an export license*". While that chunk was obviously not part of the two other possible licenses (BSD and GPLv2), export regulation compliance is license agnostic, so in doubt...

bapt

Interesting. Regarding the governance strictly speaking, The OpenBSD project seems to suck: kind of dictatorship? One guy decides for every volunteers? And what would happen if he resigns?

aja

Well, that's a bit of an exaggeration don't you think? Obviously, one guy does not decide for everything, most of the decisions are taken collegially. That said one guy, namely Theo, does have a veto over the entire project. From my point of view, this is good, it speeds up the decision process, gets things done, prevents stalling and make sure we have a safe-guard. We really can't afford losing ourselves in over-administrative processes which might lead to no decision. It is true that we like hacking more than getting ourselves caught into committee-induced compromises.

Regarding the future of the project, the only thing I have to say about it is that we are fully open source so I wouldn't worry about it. Maybe something is planned, maybe not… Look at the longest living Linux distribution as of today: Slackware; it's been run since the very beginning by a benevolent dictator and is still alive! I think it's safer for anyone, be it a regular user or a company to use OpenBSD than any company-baked project; look at what just happened to Oracle Solaris… So regarding the future of the operating system, I'd worry much more about using products from such companies than OpenBSD.

bapt

Well in the end it all depends on developers for us, so not quite different than you, core@ is elected by them every 2 years so core have confidence from the developers to drive the project, take final decision if there are no technical agreement among developers on a given subject, the team also have to deal with disputes among developers and/or community and for the project policies. All core decisions are taken over votes inside the team.

Last thing core is responsible for voting on developers. On this in particular, core focuses on source committers and delegate ports commit bits to the portmgr team and docs commit bit to the doceng team.

aja

Wait… What? You have different types of commit bits? You guys really love making things more complex than needed.

“DING!”

Scene 9: Project organisation

aja

OpenBSD has only one true commit bit. You’re a developer (with commit access) or you’re not, period. That’s good because it encourages people to contribute in all parts of the tree, not just their area of expertise. The project is comprised of 4 repositories: src (base system), www (websites hosted by the project), xenocara (Xorg with a BSD build system) and ports (ports tree).

Since every commit requires at least one explicit OK from a fellow developer, we’re pretty safe with regards to people committing in area they usually don’t. The policy is a bit more relaxed in the ports tree where we only require an explicit OK for newcomers or when importing new ports, making invasive changes or similar. Of course since you own what you commit, if you made a mistake and did not have an OK from anyone, be prepared to suffer the consequences...

bapt

We have split the repository into 3 parts: ports, base and docs when doing that we have split the commit bits as well. But note that the difference is only administrative because technically as soon as one has an access he can commit to all the repositories. This separation is more to enforce a policy, if one has been awarded a commit for docs it is because of the good work he has done on this side, the committer. He is not supposed to commit in other area without a format approval from a committer that has a bit in that area. And usually after a bunch of good patches he is granted easily with a commit bit in that area. The barrier is very soft.

While we are on administrative, I always wondered what is the process of becoming an OpenBSD committer?

aja

There is no real defined process. OpenBSD is a meritocracy so if you keep sending good patches to the mailing lists for a while, there is a good chance one developer will contact you and ask you whether you’d like to join the gang. At that point, it usually means that you’ll get invited to ICB (our internal chat) where you will get introduced to other people. The way it works then is that the developer who invited you becomes your de-facto mentor and is responsible for teaching you the few things about our development workflow and commit process etc. At that point, you get your OpenBSD account pretty fast.

That’s pretty much it, there is not “mentoring period”, your mentor is only here to get you started and take responsibility if you break something in the few weeks after getting your commit bit…

bapt

On FreeBSD when a contributor has sent a set of good patches, usually a committer gets pissed of with committing those patches he generally proposes the contributor to become a committer. If the contributor is interested, the committer sends a formal request to the core team

for base, portmgr for ports and doceng for docs. The members of the team will analyse its contributions, take a vote, and if granted will ensure the now new committer gets mentor(s) to drive him into the project: policies, knowing what resources he can access to, who is responsible for what, ensure that he probes the proper developers (the experts of the areas) for reviews before committing. When the new committer is confident enough he is released from mentorship.
Speaking of commiters and contributors, can you tell us a word about leading edge development in OpenBSD? Do you use branches or external repositories which are then merged into the main tree or?

aja

As a general rule, all OpenBSD driven software projects development is done directly in CVS. Like OpenSSH, OpenNTPD, OpenSMTPD… We do not use branches because, well, CVS.
Fun anecdote: I think we are one of the oldest and biggest running project still using CVS. I’m not saying I am proud of it, I just wanted to mention it :-)
Anyway, when invasive changes need to be done we usually make a patch available to other developers for review and the integration is done incrementally (by splitting the patch into smaller chunks). Of course sometimes it may end up being hard or next to impossible to do that in which case we often distribute snapshots including the uncommitted bits for wider testing until we are confident we can integrate that one big “patch” into the tree.
There has been a few exceptions in the past where a particular piece of OpenBSD software was developed outside of our tree. OpenSMTPD is such an example. But after a couple of years struggling with having to manually merge changes one by one “the OpenBSD way”, development went back into the tree which greatly helped general integration.

bapt

All the FreeBSD development happens in the ‘head’ branch, on the subversion repository. For big features people often create dedicated svn branches. Lots of people are also using the git forks of the tree, works on their branches there and pushed back the changes into FreeBSD via small readable chunks via git svn.
We also have a perforce repository but almost no one uses it anymore and was mainly used before the svn/git time due to CVS nightmare.

The only case where one can see very big commits instead of smaller changes are branch merges, merge from p4 (which hasn’t happened for a while) on importing new version of externally maintained softwares.

Once the code has hit the head branch if needed or willed than it is merged into the stable branches.

Before each major BSD conferences, the project organize dev summits usually 2 days. It is a unique occasions for developers to meet and to be able to discuss in live issues they faced, discussing how to make a new feature happening or project directions. Aside from those devsummits, there are some other devsummits organised around the world sometime very small sometime large. Those devsummits often turns into some kind of hackathon, some time are only in the hackathon form.

aja

It is my personal opinion that the reason OpenBSD is successful despite its relatively small amount of developers is because of hackathons. Considering our distributed development model, it is a requirement. The project is sort of a pioneer in that regard, we had regular hackathons before they became a “thing”. You can see it as a gathering of people coming from all over the world, with their own different background but with one common goal: improve OpenBSD in all area. In reality I see it as a bunch of friends coming together to have fun doing what they like to do. That’s a very important aspect I think and is what makes our developer community so vibrant and passionate. There is always some yelling, shouting, laughing, crying, despair, hope and desilution… it’s software after all. But as we like to say: “it’s not torture if you do it to yourself”.

Anyway, it’s extremely intense and motivating: it’s just amazing how many things are achieved during these events.

Hmm, I am starting to bore myself after all this talking… Should we take a break?

bapt

Yeah you’re probably right, let’s go back on the technical side after a pause.

“Interlude!”

ACT II

"Round two, fight!"

Scene 1: "BSDification"

bapt

You guys claim a lot about license purity, but you suck on that area, FreeBSD is very close to provide a base system with only permissive licenses (mostly BSD). The only few components left (for tier 1), progress can be followed here: https://wiki.freebsd.org/GPLinBase.

aja

Before we go further, could you explain to me what "tier 1" mean exactly?

bapt

Tier 1 are the main supported architecture for now i386 and amd64, various arm are close to pass in the Tier 1, It means we do ensure binary packages, binary security fixes for those arches. In term of support from source there is not much difference.

aja

On OpenBSD, if a platform is listed as supported, it's at the same level as amd64 / i386 (for the base system at least; ports is usually harder even if we try very hard to fix and support non mainstream architectures). There is no concept of tier platforms for us, we support an architecture or we don't.

bapt

The reason I talked about tier 1 is because most of the tier 2 platform are not buildable with LLVM/Clang meaning they have the old gcc/libstdc++ which means they have more GPL softwares.
Basically we are very pushing on having a full BSD toolchain. Since FreeBSD 10.0 we are using LLVM/clang as a main compiler, libc++ as C++ stack, libcxxrt for the C++ runtime. We also develop and maintain the BSD license elftoolchain which replaces most of the tools provided by GNU binutils, and is built around our BSD licensed libelf. With the main exception of the linked, which will be most certainly replaced by LLD in FreeBSD 12.
Other left non BSD components where GNU texinfo which we removed in FreeBSD 11.0 replacing the useful info pages with proper manpages (the only ones were the binutils one).
We removed GNU CVS in FreeBSD 10 and GNU rcs will be removed from FreeBSD 12.0
GDB is slowly being phased out replaced by LLDB.
So for now really the left components are:

- GNU diff (note that sdiff has been replaced by a mix of the OpenBSD and NetBSD one + patches for GNU diff compatibility)
- GNU roff while mandoc is now used to render manpages, some manpages still requires a full roff toolchain, we are going to replace it with heirloom doctools
- GNU grep (BSD grep is still too buggy but already imported)

Which is why we have hope that FreeBSD 12 will be the first release without any GNU components which is good users as well as they will benefit being able to simply use the modern version of those tools instead of the old GPLv2 version.
If we speak about tiers 2:

- gcc/libstdc++ is still required
- Dtc for embedded, beside the fact we have a BSD licensed dtc which is not yet 100% fully compliant but mostly usable.

aja

Besides a few exceptions, most of our userland is BSD. We still have (some fork of) GNU CVS, although our RCS has been rewritten a few years ago and is under a BSD license. Mandoc takes care of all our man needs, I am not aware of anything in base that would require a full roff(7) toolchain.
Regarding the toolchain you are clearly ahead in that regard but we finally did jump on the bandwagon thanks to the arm64 architecture. I will not go too much into details because that effort is somewhat recent but what I can say is that we are going pretty much into the same direction as you are (lld, libc++ etc). Of course, we still have to maintain somewhat “exotic” architectures that are not supported by LLVM/Clang and as such our old (<GPLv3) GNU toolchain is not going to go away anytime soon.
In the subject of how each of our base system is evolving, what’s the deal with that libxo(3) thing?

“DING!”

Scene 2: Over engineering

bapt

Well libxo is actually very useful. Have you ever tried to parse the netstat output from scripts? Having a programmatically parseable output is very handy to write software on top of FreeBSD quickly.

aja

That sounds like a vendor requirement... But anyway, even if I’d agree with you, does ls(1) really need to have a libxo(3) output? That looks like over engineering at its peak. All scripting languages provide API to be able to deal with filesystems so I don’t see the relevance here.

bapt

I agree that this one is arguable but it is actually somehow useful to deal with BSD extensions like chflags, where most of the scripting languages are not aware of.

aja

I am kind of torn about this. It’s not because one can implement something that one should.
Oh and what about this one: “ls -,” really?

```
   ls [-ABCFGHILPRSTUWZabcdfghiklmnopqrstuwxy1,] [-D format] [file ...]
```

bapt

Well. For this one I have no arguments :)
The fun thing is I discovered the options when looking at a slide in last year EuroBSDCon and we all thought it was a typo :)

aja

See from my perspective, this all looks very over engineered. I mean even just for the jail administration, there are probably 10+ third party tools to manage them. Why? Why can't you include a good one, instead of ten written by random people and of different quality? There is similarly a huge list of tools for controlling different aspects of ZFS, some in base and some not.
Oh and sorry for mentioning this again but: three firewalls... It can be very confusing to know which tool to use, and whether you can mix them or not especially when none of them are enabled by default. From an outsider, IPFW should be the native firewall (it was written by FreeBSD for FreeBSD) so why keep IPFilter and PF (which as far as I know has not even been synced with the upstream code in years).

bapt

For jail the default provided tools base are good enough to manage them, they are flexible and nice enough. All external tools are mostly providing extra features like provisioning etc which I don’t think belong to base, because first they can evolve faster if developed available from ports (remember a branch will live for around 5 years). Concerning ZFS, the only one I can have in mind is beadm, this one will probably end up in base after a huge rewrite.
Concerning the firewalls, true ipfw has been developed by and for freebsd, but others are willing to maintain ipfilter and they do. Concerning pf while it hasn’t been synced that is true, it is still actively maintained and have many features not available on OpenBSD: SMP, VNET for example.
Remember it is all about flexibility, and yes flexibility have a price, it can confuse users.

aja

In OpenBSD we like simplicity. I agree that we may miss a few feature here and there sometimes but we are ready to accept if it keeps us clean, simple and self-contained. It is simply unsustainable to keep accommodating “choice” and in general, complexity is the worst enemy of security which is one of our primary goals.

bapt

Flexibility does not necessary mean complexity, we also like simplicity.

“DING!”

Scene 3: Storage & filesystems

aja

Let’s not talk about that, I don’t think anyone is interested.

aja moves onto the next slide

WTF…

aja moves onto the next slide

ERR, what’s happening!!!???

aja moves onto the next slide

Oh well OK, OK. I suppose we can say a quick word about filesystems…
You go first!

bapt

FreeBSD supports multiple filesystems. The native one being the original UFS but one that have evolved to fits modern time: supporting journaling, soft updates, trim etc.
We also of course do support very modern filesystem as ZFS as a first class citizen.
Other filesystems we do support are msdos, ext2,3,4 (without journal), nandfs.

aja

On OpenBSD we are stuck with the traditional BSD file system: UFS/FFS. That said, it has been extremely stable for us. After 15+ years of OpenBSD in production in all kind of different setups, I have never lost a file.
We obviously also support NFS but only version 2 and 3 since version 4 is a complete rewrite requiring pieces we really do not want to support in base (like GSS-API). Our automounter is still amd(8) from 4.4BSD! It’s not very efficient but it gets the job done. I for one would welcome a more modern implementation…
Regarding Internet SCSI, we have a native iSCSI initiator implementation: iscsid(8).
I would not argue that file system is not really where we shine, we have not journaling, we do have soft updates but we still have some bugs in that code pah, like your system could panic if there is an unrecoverable I/O failure. Note that a standard OpenBSD installation does not enable softdep on any file system.

bapt

Wow you still have bugs with softdep???

aja

Yeah but that’s not fair, you guys have Kirk (McKusick)! It’s hard to compete! ;-)

bapt

Under the hood we have the GEOM framework which is very powerful giving us support for multipath (gmultipath, encryption (geli, gbde), mirroring (gmirror), network transport (ggate) or even faking the hardware with gnop.

We have a very good support for iSCSI targets (via ctld including HA) and initiator (iscsid).

On network file system we have very good support for NFS (all versions) server and client.

We provide native tools to deal with storage devices: SES, SAS cards etc.

As a client of storage we also have the traditional amd(8) and since recently we have a shiny new autofs support (automountd).

And that is only speaking of user facing features and utility.

aja

Well I agree that GEOM is generally a better solution maybe except with regard to multipath which we do at the SCSI layer with mpath(4): it doesn't require manual configuration, you just plug it into capable hardware and it figures it out.

Regarding ZFS, it's really a kitchen sink… Is it a good thing? Is it bad thing? I honestly don't have a strong opinion on the matter so I don't know. It is a very nice piece of technology and comes with some very nice features, no argument here. However the fact that it strongly recommends using ECC memory worries me a bit.

Software RAID support is generally in a good shape on OpenBSD except for the RAID5 discipline on which we cannot boot from… I agree that it's a bit embarrassing.

bapt

About encryption we have multiple choices around the GEOM layer:

- geli
- gbde

Because they are on the geom layer, then we can put any filesystem on top of it.

For geli since recently (FreeBSD 11.0) we can support full disk encryption decrypted by the bootloader using a passphrase.

We also support (not in base by developed for FreeBSD) filesystem level encryption via pefs which allows to encrypt only some directories whatever the filesystem is under the hood.

aja

The way we do disk encryption on OpenBSD is similar to how we manage RAID devices (software or hardware) by using the bioctl(8) management interface utility over a softraid(4) encrypting discipline virtual device. So it is already familiar to people and the process is very simple (just one command really). We have support for full disk encryption, decrypted by the bootloader either by using a passphrase (interactive) or a "keydisk" (ideal for non-interactive use, using a USB dongle or similar).

bapt

By the way swap encryption on FreeBSD is also very simple just use geli or gbde.

aja

Our swap has been encrypted by default for many years without the need to do anything. It's my opinion that swap should always be encrypted. Your SSH or PGP private key could actually end up there at some point… There is a sysctl(8) to disable swap encryption if really needed. I am mentioning it because that is an exception to the general OpenBSD rule where security enhancements are enabled by default and can hardly be disabled.

"DING!"

Scene 4: The base system, part II

aja

Speaking of "default" settings, that is really something I love about this operating system: they make sense and are very well thought out. We regularly have lengthy arguments about what feature should be available by default and in which in way it should work out of the box. That's how we came up with so many simple daemons.

The network related ones are a very good example because they solve hard problems in simple ways:

- vxlan, relayd, bgpd, rtables, ospfd, carp, ipsec, spamd, httpd, dvmrpd, iked, dhcpd, acme-client (Let's Encrypt)...

I've seen people understand how networking works by just reading the configuration and man page of OpenBGPD and similar.

bapt

While it is true that OpenBSD is full featured in the network side FreeBSD is also very very full featured:

- vxlan, carp, FIB, vimage, netmap, ipsec, netgraph

Back to the base system, that makes the base system huge maybe some of those tools could be just installed via ports why having them in base?

aja

We consider OpenBSD a general purpose OS that provides a useful number of services out-of-the-box. It's an important design decision because it means that all these tools are developed together. A change in the kernel or a library will immediately trigger some modification to these daemons when need be. Also, anything that is part of base is audited, pledged and follows our standards; the situation is totally different in ports.

Also it encourages code synchronisation and sharing amongst the base system. That would be a totally different thing is these were part of the ports tree: runtime breakage may become unnoticed for some time…

I for one am very happy to have so many features in the base system and the funny thing is that OpenBSD base system is still smaller than of FreeBSD.

So it is quite surprising that you speak about a huge base… On one end you strip down your base system (removing texinfo, perl, etc) but on the other end you have three firewalls.
We on the other end like to concentrate on having one way to do things.

bapt

The base system for us is a coherent out of box general purpose system. For us a full feature version of given server can be installed from packages and there is no reason why we should bloat the base system with those tools. Having a full featured MTA in base is almost impossible: how one would add the feature required via tons of external libraries which are not available in base (ldap connection, antivirus scan, etc). Why would I bother having an http server installed on my storage server? Of course if a FreeBSD developers maintains that code, then there is no need to remove it, that is how we ended up with 3 firewalls. With exception of course of servers closely tied with the kernel like iSCSI target (ctld), nfsd, etc. But in that case we do provide KNOBS so one can build a stripped down version of FreeBSD if they need it.

aja

I think that is a major difference between our operating systems: we do not want to provide knobs for everything, we want to provide what “we” consider sane defaults and best practices to enforce “our” ways. Also, is it the job of the operating system to provide embedded and/or stripped-down “editions”?
An OpenBSD base installation is basically a complete fork kit in itself, meaning you have everything you need to continue developing the operating system (utilities, compilers, distribution tools, CVS, source code, etc). Similarly to FreeBSD, the non-BSD/ISC licensed parts are isolated in the source tree so you can easily and safely make commercial products and appliances from OpenBSD without putting the responsibility of providing build knobs on us.
Talking about knobs, doesn’t that make FreeBSD more a toolkit to build an operating system than a proper one?

bapt

No FreeBSD by default is a full featured OS and on the binary form we do only support the plain full FreeBSD as served by the ISOs. but keep in mind that FreeBSD is also widely used in embedded environments or inside appliances, both needs lots of flexibility (either stripping down the size for embedded, or being able to replace one feature we do provide by their own implementations). Also we do encourage a lot vendors to work closely with their upstream meaning us.

aja

Yes FreeBSD encourages vendors to contribute and commit: it wants to be a “universal” operating system like Linux and satisfy the maximum amount of people. If you look at it from a different perspective, you could also say that FreeBSD does what your employer wants while OpenBSD does what its developers want.

Anyway, I like the fact that I can have “functional problem solving box” in no time without the need to install any third-party packages while staying very small in size. For instance I don’t need to install a proper shell which I can work with… I am not talking about a heavily featured shell ala zsh, but “/bin/csh” as the default root shell you guys… really?

bapt

If someone is willing to push a new feature and/or a new KNOB to improve flexibility as long as it does not break FreeBSD directions and usability we will accept it. That gives us lot of flexibility in the base system and is very nice to make freebsd usable for very niche usage. For example making a very thin storage only server on ramdisks (like we do in Gandi).
Which makes me think I see that OpenBSD does not support kernel modules, I find those very handy for flexibility as I can load or unload features on demand if I need it without having to rebuild the kernel we also have sysctl and tunables to be able to interact with the kernel configuration. Isn’t it painful to tune your kernel?

aja

While it’s true that we removed support for loadable kernel modules, we still have dynamicity thanks to our config(8) utility. It’s used to modify a kernel without recompilation. Device parameters that are usually hard-coded in the kernel can be changed, added, removed… It does not allow you to inject new code into the kernel but it gives you the choice to enable or disable existing code. To give you a “Joe User” example, it’s often used to disable ulpt(4) (USB printer) support to make one’s printer seen as a regular ugen(4) device usable by libusb.

“DING!”

Scene 5: Virtualization & running alien software

bapt

On FreeBSD we do support running linux binaries natively via the linuxulator or linux emulation. Right now we do support both linux i386 and amd64, there are patches to support linux on arm.
Also we do support running old FreeBSD binaries on newer FreeBSD meaning one can run FreeBSD 4 userland and binaries on FreeBSD 11, inside a jail if they need to for whatever reason.
On the virtualization front we are very well featured, we do support native virtualization via bhyve
It has many nice features like netmap iface support, vnc server and can run almost any OS as guest as long as it supports virtio.
But we also support external virtualization mechanism: xen dom0 and domU (natively), virtualbox (via ports).

aja

Well, having Citrix people working as developers does help having Xen Dom0 support. I think it boils down to the fact that you guys made the decision a long time ago to not fuck with the hardware as much as possible but let hardware manufacturers do it and instead concentrate on something else. On OpenBSD, we prefer to try and convince the vendors to open their specifications to let us write our own drivers. Your approach is probably more "pragmatic" but as you probably know by now, we like to be in control of our operating system. Not everything is black and white of course so this is to be taken with a grain of salt since there is cross-pollination between our projects and we do use some parts of your vendor-written drivers (ix(4) for example).

bapt

Being open to vendors is in my opinion a pragmatic point of view and does not make us lose control of our sources actually we have really good relationship with plenty of them, their committers running through our usual process to recruit new developers (mentorship etc). We review the code they commit, and even extend it sometime. It is not unusual to see vendors committers continuing to work on FreeBSD after they stopped working for the said vendor.
But back on virtualization what is the current situation on OpenBSD?

aja

Let’s start with vmm(4) since it’s been mentioned a couple of times already. Currently it can reliably run OpenBSD, NetBSD, some Linux guests and provides a very friendly user interface with vmclt(8).
Since it is chrooted and pledged, breaking out of the VM monitor means ending up in a chroot with a very limited list of allowed syscalls… not much one can do.
There is also some ongoing work to be able to have qemu(1) working on top of vmm(4) mostly to take advantage of existing VM management tools; but there is nothing concrete yet.
The other virtualization technology that OpenBSD supports is sun4v logical domains (LDOM). It’s hardware virtualization (i.e. the resources are partitioned directly at the hardware level), is supported on SPARC V9 processors and is very secure by design since there is no software “hypervisor” involved: the processor itself runs in hyper-privileged execution mode. It can natively run any sparc64 operating systems (Linux included).

“DING!”

Scene 6: Out of the BSDs

bapt

Right ok, so we speak about our project internals, but what about the relationship with external projects.
In general for us most upstream are happy to receive FreeBSD related patches and integrate them even big ones.

For example dealing and upstreaming patches with LibreOffice is very easy they are very welcomed and even tried to setup a CI based on it.
On the GNOME/GLib front while I don't deal with them much we have very good relationship with the GLib maintainer who came to us to ensure GLib is BSD friendly he even has set up the GNOME CI on FreeBSD to ensure everything builds as fine as possible.

aja

We share a same experience I guess. While some upstreams are more opened than others to non Linux contributions, in general they are happy to integrate our patches (and sometimes learn a thing or two about portability in the process).
It's true that a few identified people within the FOSS community takes some pleasure in bashing anything that is not GNU or Linux but they are far from being the norm, fortunately for the ecosystem!
Sometimes, upstream can be very close, we have OpenBSD developers who are also developers at GNOME, XFCE, Mozilla, LibreOffice... And the opposite is true with OpenBSD being the upstream for several software like OpenSSH, PF, OpenSMTPD, LibreSSL, mandoc…
Some applications can be very complex to port so it's important to be able to interact nicely with upstream. And when things start crashing, well, that's when our debugging tools come into action!
I will not go into the details of such tools which are probably similar to yours, I'll just mention that we are building the entire base system with debug symbols ("-g") and that we are thinking about doing the same for ports to provide different sets of packages (regular and debug ones).

bapt

FreeBSD is pretty well stuffed for debugging. By default in base we have ktrace and truss to be able to track what an application is doing. In particular when debugging a capsicumized application ktrace is able to tell the developer what capability his application is missing if any.
More importantly we now also have dtrace in base, which is a very powerful tool to be able to debug. When building the FreeBSD kernel for example all debug flags are converted to the CTF format, so are dtraceable.
To help debugging we also build by default the base system with debug flags (-g) but because we do care about embedded systems we extract into a separate file. Lldb and gdb are configured to know where to look for them.
Concerning the ports tree, now lots of application are dtrace aware, but the ports tree itself is not built with debug flags yet, but there are work in progress to be able to extract them from the binaries just like in base and provide -debug packages.

aja

That is yet another area that has seen a lot of development lately. Our ddb(4) (the kernel debugger) just got basic support for CTF. No that has nothing to do with Capture The Flag...
Once being included into the binaries, CTF (or Compact/Compressed Type Format) will provide a subset of the information from DWARF debugging sections like definitions of data type and functions used by debugging tools.

We also have a ctfdump(1) and ctfconv(1) implementations and we can dynamically activate kernel profiling using DTrace-like probes. So we have the bedrock for DTrace on OpenBSD.

bapt

It looks like you guys are getting there which is great. So both projects offer nice debug capabilities, now what we both need are more contributors! The tooling is here!
Anyway, since we mentioned GNOME and GLib, let’s talk about how we shine (or not) on the Desktop and multimedia side.

“DING!”

Scene 7: BSD as a Desktop

aja

Sure, I’ll start with audio if you don’t mind…
On OpenBSD, system audio and MIDI support is handled by the sndio(7) programming interface. Just like FreeBSD OSS, it supports device sharing, conversions, resampling, per-program volume knob and most features that modern systems have. However, while OSS is a large pure kernel implementation, sndio(7) is built around a small user-space daemon (of the size of getty(8)): no code does signal processing with kernel privileges but instead it is done using the unprivileged “_sndio” user. It is comprised of 17 functions (sio_xxx) that do one thing and while being relatively small, it gives you network transparency, per application audio mixing and virtualization (audio devices and MIDI ports are available on the network transparently which allows virtual machines to use the host soundcard for example), record-what-you-hear device, MIDI ports virtualization... In a word, it follows the Unix and KISS philosophy.
If I am to be honest, there is one thing that is still missing from sndio(7): mixer support; we still need to use the old SunOS API for that. Also when it was first implemented, it did require some porting effort in ports compared to OSS (pre-2002 default Linux API that a lot of programs already supported) but as one says: "no pain, no gain" and as far as I know, OSS still has no way to obtain a device block size not to recovers from underrun and overruns.

bapt

FreeBSD has a full-featured sound subsystem, giving low-latency in-kernel mixing allowing multiple applications to play sound (with independent volume controls enabled by default) with no configuration.
It has an OSS compatible API from userland perspective. Which makes (made) porting code to it very easy.
It has multi channel audio support enabled by default (It can go up to 18 channels but is currently set to 8 - which correspond to 7.1 surround sound), software equalizer.
About the device block size, there are mechanism in OSS for that it was even in the old linux implementation, via SNDCTL_DSP_BLKSIZE (which we do support on freebsd) along with our

old AIOGSIZE. In case an application does not take that into account (by getting first the block size) a user can workaround the issue by using a sysctl: hw.snd.latency.

The only thing we lack here is network transparency where because sndio works on FreeBSD as well, we can use your stuff to have this feature. So we can have all our native features and also benefit all your features for the price of simply installing the sndio package.

aja

Now, regarding the Desktop per se, we do have some limitations and as far as I know, you guys have similar ones. The first one would be wireless support. While there are lots of differences in supported features and hardware (different 11n rate scaling implementation…), I agree that FreeBSD is now a bit ahead of us in that regard. A few years ago, when Damien Bergamini was frantically working on WiFi, it was a different story and I believe you guys did base some of your WLAN drivers on ours. Anyway we are slowly closing the gap though but we are still missing support for bwn(4) and mwl(4) as well as Tx aggregation and 40 MHz channels. But even with or without it, I think both projects do lag a bit behind in general when it comes to supporting recent WiFi hardware.

bapt

Well the wireless situation on FreeBSD is not really better on FreeBSD than on OpenBSD I'm not even sure we have more people on it than you do :). Often our drivers are first imported from OpenBSD and then adapted and expanded.
Another issue is graphic drivers, we are always lagging behind with very few people actively working on the kernel part. Meaning in FreeBSD vanilla kernel we can only support up do haswell GPU which is… We have an external repository where some people are actively working on bringing newer drm drivers to FreeBSD with that project one can run FreeBSD on nearly all intel chips but this is far from being mergeable.
So I would say that for US it sucks.

aja

Yes indeed, graphics support can also be problematic for us; it's again mostly due to only having a handful of people working on it. I'm actually impressed by the work they've been doing. But while we make a good job on having X.org and the userland stack up-to-date (version 1.18.4 at this date), we cannot always track the most recent changes or features requiring kernel assistance. I think on that topic we are pretty much in the same boat.

bapt

On the userland part we are not that bad now. We are on xorg 1.18.4 as well aka almost the latest, we have finally made our duty and provided a patch to the xorg configuration mechanism so it can directly detect inputs drivers and hotplugging via devd instead of relying on HAL (something you did long ago). We also now have evdev support in kernel (will be in FreeBSD 11.1) which makes it even easier to deal with input drivers.

Recently we have gain wayland support, it has been partially committed it, it is usable as a library and via the scfb driver on EFI with vanilla kernel. For fully accelerated one, it requires features only available for now on the external project I talked to you about.
What is your situation regarding wayland? And btw can you explain me how you deal with xorg? I heard of xenocara, is it a fork of Xorg? Is it part of the base system?

aja

Xenocara is not a fork of Xorg. In essence it’s a BSD build infrastructure used to natively compile and provide Xorg (and a couple of other graphical projects like fvwm or cwm) on OpenBSD. Xenocara was born at the time Xorg was split into multiple different modules and I think it was because it was the easiest way to maintain it on OpenBSD while keeping upstream hierarchy and hence ease “up and down” merging of code. Also remember that our X(7) server has been running with privilege separation (using a dedicated _x11 user) for ages.
Anyway, Xenocara is not part of the base system per se but is part of the default OpenBSD installation. It’s available in the same way as our base system sets, there are no X packages. You can see it as in-between ports and base. The important thing is that it is assumed to be found on a standard OpenBSD installation which is an important fact for packages(7) for example.

bapt

I see. By the way, since we are talking about the Desktop and by extension, the user experience, how is your localisation support?

aja

Regarding character encoding support, the OpenBSD base system deliberately supports US-ASCII and UTF-8 only. It allows for much better error handling and much more stable operation than a system supporting arbitrary character encodings where a single damaged byte in a text file often results in all the rest of the file to be unreadable. It is required that the system is UTF-8 only, or you don't get the benefit. I believe that on FreeBSD, if there is a single invalid byte, cut(1) loses the complete rest of the input file even when you use LC_CTYPE=en_US.UTF-8. Because ASCII is compatible with UTF-8, we can enable UTF-8 by default in most programs while making the terminal safer and more resilient against control character attacks. A visual example of the differences is pod2man(1) that is UTF-8 enabled by default on OpenBSD, which FreeBSD cannot do because they support arbitrary locales. Consequently, non-ASCII characters work in Perl manuals on OpenBSD, but not on FreeBSD; compare "man perlunicook" on both systems.
Of course, it's OpenBSD so there is also the question of security. If terminals are allowed to use arbitrary encodings, no text in any encoding can be safely displayed, because depending on the encoding, anything might be a control character. With UTF-8 only, ASCII is always safe for display.
To summarize, FreeBSD attempts general locale(1) LC_* support. OpenBSD deliberately does not, completely ignoring LC_COLLATE, LC_MESSAGES, LC_MONETARY, LC_TIME,

NLSPATH, and in particular LC_NUMERIC. I hope that you guys are not going to implement the insane non-standard GNU extensions (LC_ADDRESS, LC_MEASUREMENT…).
Speaking of which, I am sure you have a nice story about implementing LC_COLLATE in FreeBSD libc…

bapt

Well on FreeBSD most LC_* were mostly abandoned for a while, I recently decided to update those and because it is a nightmare to deal with all the individual encodings I decided to revive a very old project we have which basically means take from CLDR all the resources we will never be able to adapt ourselves and generates those files from there. Along with adding unicode collation support to our libc which is a nice story of collaboration between FreeBSD, Dragonfly and Illumos but that's another story.
It is really great to have unicode collation in particular for database indexes and more.
A fun side effect is that apparently no one expect what unicode collation means and that ASCII is not the only characters available. So sorting in a locale aware unicode world results in a behaviour which can be different from what people may expect. In particular for basic unix tools like tr and friends, where [a-z] does not mean the same as in ASCII.
All our base tools are locale aware but given we did not have unicode collation then no one noticed that, since we have unicode collation people tends to get surprised. On the GNU world they “fixed” that making those tools not locale aware. I prefer to make people learn about classes which were made for that: [:lower:] for example.

aja

Let’s go back to our current topic: the Desktop. Regarding the GNOME Desktop environment, a huge amount of work has been done to port and maintain it because we had a need to deploy and maintain a few thousands OpenBSD-based desktops. I think we’ve been ahead in that regard for a few years until you guys catched up thanks to the FreeBSD GNOME team. We actually collaborated quite a lot and I am still wandering on the freebsd-gnome IRC channel! I really appreciate when we are able to work together for the greater good of the BSDs in general. Besides that, it’s surprising but OpenBSD actually supports most “fashion desktop computing” softwares like DBus, ConsoleKit, PolKit, chromium, Firefox, LibreOffice, KDE, XFCE, CUPS...

bapt

About desktops we are pretty much up to date on GNOME (mainly maintained by Koop Mast whom collaborate with you for upstreaming), KDE, xfce, mate are available in their latest version as far as I know. Firefox is well maintained (probably more contributors would help) the nice thing about Firefox is the collaboration between the FreeBSD and the OpenBSD folks. LibreOffice is also in pretty good shape very few patches necessary. The main issue I can see in ports now is probably Google chrome, while it works and I have to express a very big Kudos to the maintainers, upstream is really unfriendly is part of the very few upstreams that actually completely ignore the BSD related patches most of the time. To have it in ports we do have to main 484 patches…. Large part of them are very easy to upstream...

“DING!”

Scene 8: Authentication, authorisation

bapt

For the authentication we do use PAM, actually the OpenPAM implementation (developed and maintained by a FreeBSD developer) which is compatible with the PAM specification. It gives us access to lots of various external modules even if for most of them we do prefer relying on our own implementation. The nice thing is it makes it simple to port common softwares as they all support pam.

aja

BSD Auth originally came from BSD/OS (BSDi) and was later adopted by OpenBSD. One of the differences between BSD Auth and PAM is that PAM "modules" are libraries which must be loaded into the application. BSD Auth "modules" are effectively separate applications or scripts located under /usr/libexec/auth/ that are run as a separate process from the one authenticating, allowing them to communicate over a simple IPC interface. That means we never expose the credentials store to possibly buggy software. It matches the traditional privilege separation model we are used to on OpenBSD while still being able to provide different way to authenticate (LDAP, Kerberos, YubiKey, RADIUS…).

While PAM may be a bit more flexible and way more commonly found (so there are a lot of different authentication modules readily available), it usually requires elevated privileges to authenticate. Applications using BSD Auth only need to be in the "_auth" group to be able to run the /usr/libexec/auth/ helpers. Just look at your OpenSSH security advisories, most of them are PAM related.

bapt

In the case of OpenSSH security advisories, I think it is also probably related to the fact that upstream is paying less attention (for the obvious reasons you stated above) but that is true that pam API is not simple to use and so easily error prone. In the OpenPAM implementation, while it is compatible with the official API there is also plenty of helpers to simplify adding support for pam in applications so that it is less error prone.

For name services we do use NSS which provides us lots of flexibility through its modular nature. Note that our NSS api are not 100% compatible with the GNU libc one.

We also provide a nscd daemon which can cache name service response (per user), but is not limited to that, It can also performs the requests which means the modules are no longer loaded into the libc but through that dedicated daemon.

aja

Regarding authorisation and virtual system users, we only have support for traditional YP/NIS. OpenBSD does not use NSS (nsswitch.conf(5)) for basically the same reasons it does not use PAM, we do not want dynamically loaded modules to play games in our libc and resolver. We do have support for getting users from an LDAP server thanks to ypldap(8). It’s a daemon providing YP maps using LDAP as a backend. It is a replacement for the traditional ypserv(8) daemon and is compatible with any ypbind(8) implementation . Actually I see that you guys imported it some time ago and I am interested why, wasn’t nss-ldap good enough or is it again to satisfy “flexibility”?

bapt

I still wonder myself. It doesn't hurt but... hey. I know at least one university that was happy to see it so they could have simpler steps to migrate to full LDAP according to one of its administrator.

“DING!”

The End

aja

So, all things considered, I think it's pretty obvious that my BSD sucks less than yours!

bapt

I agree to disagree on that one, it's clear my BSD sucks less than yours!
That said, I think there are areas where we suck equally, like wireless or display drivers...

aja

That is very true, we both suck on some aspects!

bapt

But as much as we like to make fun of each other, we are not only sharing bad things.

aja

Indeed, I think cross-pollination between our two operating systems works quite well. We actually do exchange lots of things and it would not make sense to list everything here.

bapt

OpenBSD has imho an important role, in the Open Source land and more, You guys are tackling very important project which would probably never have happened otherwise. The most famous one that comes to my mind is OpenSSH, I really like how open you are to portability for those softwares given the amount of extra work it requires to do that: OpenSMTPD, tmux, mandoc, sndio are very good example of that. Often teaching upstreams about good (and secure) coding practices.

aja

I think FreeBSD is important in the global ecosystem. It's a real "enterprise" operating system and I think it is slowly filling the spot left by Solaris. It bundles some amazing pieces of technology and in some area is still on the edge of innovation. Some very large entities use it and thanks to FreeBSD, a lot of people have been made aware of the BSD community in general.
For me it is a good weapon to make people aware that "fringe" operating systems are certainly not lagging behind Linux.

bapt

For me in the FreeBSD is a wonderful Operating Systems very flexible making attractive for almost all use cases. The project is very open, and everyone from vendors to individual have

their place, while there are lots of vendors that contributes to FreeBSD, the project remains completely community driven and individual can easily find their path in the project.
As an example in less than 2 years I have been able to bring very important modifications in the project and drive lots of the directions the project has taken.

aja

We are a small project but I am proud when I see that in some areas a small amount of hackers can compete with a huge project like FreeBSD and sometimes deliver things ahead. We do serious things without taking ourselves too seriously.
You guys say our performance sucks, we say your security sucks. I suppose there is some truth in both stereotypes.
I see OpenBSD as some kind of incubator and bedrock for new technologies that is not afraid to break things; sort of a destroy to build approach.
I encourage people to try it out as a power user or developer. Not just install it but really try using it: I have been surprised at how many people actually have a misconception of OpenBSD even within the BSD community itself... In my experience, besides the obvious benefits OpenBSD is known for (proactive security and all), it has been one of the easiest and best documented operating systems I have ever used. I see "simplicity" as an art form, we are trying very hard to prevent introducing a large amount of maintenance and complexity overhead for solving theoretical problems in usage cases that are not real; if you ever feel the need to overtune or harden something, then you are clearly on the wrong operating system...

aja+bapt

Thank you!

“Final DING!”