

Easier Python
paths with pathlib

Write Secure
Shell Scripts

Free Software,
Open Science and R

LINUX JOURNAL

Since 1994: The original magazine of the Linux community

THE SECURITY ISSUE

*The Heads Project:
a Free Software Solution
for Secure Booting*

YubiKey 5 and Web Authentication

*The New Purism Librem Key
Hardware Token*

Password Manager Roundup

De-mystifying X.509 Certificates



62 *DEEP DIVE:* **Security**

63 **Password Manager Roundup**

by Shawn Powers

If you can remember all of your passwords, they're not good passwords.

80 **Everyday Security Tips**

by Michael McCallister

Make your computer safer with these guidelines based on the Linux Foundation's Security Checklist developed for corporate systems.

91 **Understanding Public Key Infrastructure and X.509 Certificates**

by Jeff Woods

An introduction to PKI, TLS and X.509, from the ground up.

104 **WebAuthn Web Authentication with YubiKey 5**

by Todd A. Jacobs

A look at the recently released YubiKey 5 hardware authenticator series and how web authentication with the new WebAuthn API leverages devices like the YubiKey for painless website registration and strong user authentication.

122 **The Purism Librem Key**

by Todd A. Jacobs

The Librem Key is a new hardware token for improving Linux security by adding a physical authentication factor to booting, login and disk decryption on supported systems.

134 **Tamper-Evident Boot with Heads**

by Kyle Rankin

Learn about how the cutting-edge, free software Heads project detects BIOS and kernel tampering, all with keys under your control.

6 The Security Issue

by Bryan Lunduke

10 From the Editor—Doc Searls

A Line in the Sand

14 Letters

UPFRONT

20 Some (Linux) Bugs Have All the Fun

by Bryan Lunduke

24 Astronomy Software by Any Other Name

by Joey Bernard

32 Patreon and *Linux Journal***33 Reality 2.0: a *Linux Journal* Podcast****34 News Briefs**

COLUMNS

38 Reuven M. Lerner's At the Forge

Easier Python paths with pathlib

46 Dave Taylor's Work the Shell

Writing Secure Shell Scripts

52 Zack Brown's diff -u

What's New in Kernel Development

178 Glyn Moody's Open Sauce

If Software Is Funded from a Public Source Its Code Should Be Open Source

ARTICLES

150 Programming Text Windows with ncurses

by Jim Hall

How to use ncurses to manipulate your terminal screen.

166 Open Science, Open Source and R

by Andy Wills

Free software will save psychology from the Replication Crisis.

AT YOUR SERVICE

SUBSCRIPTIONS: *Linux Journal* is available as a digital magazine, in PDF, EPUB and MOBI formats. Renewing your subscription, changing your email address for issue delivery, paying your invoice, viewing your account details or other subscription inquiries can be done instantly online: <https://www.linuxjournal.com/subs>. Email us at subs@linuxjournal.com or reach us via postal mail at *Linux Journal*, 9597 Jones Rd #331, Houston, TX 77065 USA. Please remember to include your complete name and address when contacting us.

ACCESSING THE DIGITAL ARCHIVE: Your monthly download notifications will have links to the different formats and to the digital archive. To access the digital archive at any time, log in at <https://www.linuxjournal.com/digital>.

LETTERS TO THE EDITOR: We welcome your letters and encourage you to submit them at <https://www.linuxjournal.com/contact> or mail them to *Linux Journal*, 9597 Jones Rd #331, Houston, TX 77065 USA. Letters may be edited for space and clarity.

SPONSORSHIP: We take digital privacy and digital responsibility seriously. We've wiped off all old advertising from *Linux Journal* and are starting with a clean slate. Ads we feature will no longer be of the spying kind you find on most sites, generally called "adtech". The one form of advertising we have brought back is sponsorship. That's where advertisers support *Linux Journal* because they like what we do and want to reach our readers in general. At their best, ads in a publication and on a site like *Linux Journal* provide useful information as well as financial support. There is symbiosis there. For further information, email: sponsorship@linuxjournal.com or call +1-281-944-5188.

WRITING FOR US: We always are looking for contributed articles, tutorials and real-world stories for the magazine. An author's guide, a list of topics and due dates can be found online: <https://www.linuxjournal.com/author>.

NEWSLETTERS: Receive late-breaking news, technical tips and tricks, an inside look at upcoming issues and links to in-depth stories featured on <https://www.linuxjournal.com>. Subscribe for free today: <https://www.linuxjournal.com/enewslatters>.

LINUX JOURNAL

EDITOR IN CHIEF: Doc Searls, doc@linuxjournal.com

EXECUTIVE EDITOR: Jill Franklin, jill@linuxjournal.com

DEPUTY EDITOR: Bryan Lunduke, bryan@lnduke.com

TECH EDITOR: Kyle Rankin, lj@greenfly.net

ASSOCIATE EDITOR: Shawn Powers, shawn@linuxjournal.com

EDITOR AT LARGE: Petros Koutoupis, petros@linux.com

CONTRIBUTING EDITOR: Zack Brown, zacharyb@gmail.com

SENIOR COLUMNIST: Reuven Lerner, reuven@lerner.co.il

SENIOR COLUMNIST: Dave Taylor, taylor@linuxjournal.com

PUBLISHER: Carlie Fairchild, publisher@linuxjournal.com

ASSOCIATE PUBLISHER: Mark Irgang, mark@linuxjournal.com

DIRECTOR OF DIGITAL EXPERIENCE:

Katherine Druckman, webmistress@linuxjournal.com

GRAPHIC DESIGNER: Garrick Antikajian, garrick@linuxjournal.com

ACCOUNTANT: Candy Beauchamp, acct@linuxjournal.com

COMMUNITY ADVISORY BOARD

John Abreau, Boston Linux & UNIX Group; John Alexander, Shropshire Linux User Group; Robert Belnap, Classic Hackers UGA Users Group; Aaron Chantrill, Bellingham Linux Users Group; Lawrence D’Oliveiro, Waikato Linux Users Group; Chris Ebenezer, Silicon Corridor Linux User Group; David Egts, Akron Linux Users Group; Michael Fox, Peterborough Linux User Group; Braddock Gaskill, San Gabriel Valley Linux Users’ Group; Roy Lindauer, Reno Linux Users Group; Scott Murphy, Ottawa Canada Linux Users Group; Andrew Pam, Linux Users of Victoria; Bob Proulx, Northern Colorado Linux User’s Group; Ian Sacklow, Capital District Linux Users Group; Ron Singh, Kitchener-Waterloo Linux User Group; Jeff Smith, Kitchener-Waterloo Linux User Group; Matt Smith, North Bay Linux Users’ Group; James Snyder, Kent Linux User Group; Paul Tansom, Portsmouth and South East Hampshire Linux User Group; Gary Turner, Dayton Linux Users Group; Sam Williams, Rock River Linux Users Group; Stephen Worley, Linux Users’ Group at North Carolina State University; Lukas Yoder, Linux Users Group at Georgia Tech

Linux Journal is published by, and is a registered trade name of, Linux Journal, LLC. 4643 S. Ulster St. Ste 1120 Denver, CO 80237

SUBSCRIPTIONS

E-MAIL: subs@linuxjournal.com

URL: www.linuxjournal.com/subscribe

Mail: 9597 Jones Rd, #331, Houston, TX 77065

SPONSORSHIPS

E-MAIL: sponsorship@linuxjournal.com

Contact: Publisher Carlie Fairchild

Phone: +1-281-944-5188

LINUX is a registered trademark of Linus Torvalds.



privateinternetaccess[®]
always use protection[®]

Private Internet Access is a proud sponsor of *Linux Journal*.



**Join a
community
with a deep
appreciation
for open-source
philosophies,
digital
freedoms
and privacy.**

**Subscribe to
Linux Journal
Digital Edition
for only \$2.88 an issue.**

**SUBSCRIBE
TODAY!**

THE SECURITY ISSUE

By Bryan Lunduke

On January 13th, 2018—at 8:07 am—an emergency alert was issued in Hawaii. The message, in its entirety: “BALLISTIC MISSILE THREAT INBOUND TO HAWAII. SEEK IMMEDIATE SHELTER. THIS IS NOT A DRILL.”

Although this message—which showed up on smart phones across the state—was, indeed, not a drill...it also was not a real threat. There was no missile hurtling through the atmosphere towards Hawaii. It turns out someone had simply clicked the wrong option from a very poorly designed user interface and sent out a fake (but very real-looking) emergency alert.

This is officially known as a “whoopsie daisy”.

As the story spread around the globe, obviously all the news reports were going to need a picture to run along with it. As luck would have it, the Associated Press had published a picture taken inside the Hawaii Emergency Management Agency—showing computer workstations where they watch for such possible threats. This picture was spread far and wide.

On that picture, people noticed something. Something amusing. Something, for many of us, relatable.



Bryan Lunduke is a former Software Tester, former Programmer, former VP of Technology, former Linux Marketing Guy (tm), former openSUSE Board Member...and current Deputy Editor of *Linux Journal* as well as host of the (aptly named) *Lunduke Show*.

THE SECURITY ISSUE

On one of the monitors was a sticky note. With the password written on it.

(There were actually two sticky notes on the monitors in the picture. The second sticky note contained the message “SIGN OUT”. Because, you know, security is important.)

While the accidental, non-real emergency alert was not caused by any sort of security breach (sticky-note-based or otherwise), this picture served as a great reminder to the entire world that we probably shouldn’t write down our passwords on sticky notes. Not even a government agency tasked with Emergency Management is immune to this sort of weak security.

It reminds me of a scene from the Mel Brooks’ film *Spaceballs*. In the film, an advanced security barrier had been constructed around a planet. The dastardly space-villains forced the king of the planet to give up the code that would open that barrier. That code? 12345. Upon learning of the code, one of the characters was shocked. “Remind me to change the code on my luggage.”

Any of this sound familiar? Perhaps it’s time to get rid of the sticky notes—and the passwords that are no more complex than “password123”—and get yourself a good password manager.

In this issue, Shawn Powers provides a good “Password Manager Roundup”, laying out the pros and cons of various options.

Then, while you’re in a security frame of mind, familiarize yourself with a good set of guidelines (based on the Linux Foundation’s Security Checklist) for how to keep your system secure with Mike McCallister’s “Everyday Security Tips”.

Following these suggestions will make you far more secure than that Emergency Agency in Hawaii or that planet in *Spaceballs*, but what if you want to take things a step further? What if you want to dive into the world of encryption and hardware security keys?

First things first: get a basic grasp on how current, modern encryption works with

THE SECURITY ISSUE

Jeff Woods’ “Understanding Public Key Infrastructure and X.509 Certificates”. It may not seem like a page-turner, but trust me. This is good stuff to know.

Then, move on to hardware security keys and the benefits they can provide to Linux-based workstations and laptops.

Todd A. Jacobs’ “WebAuthn Web Authentication with YubiKey 5” gives an overview to using a YubiKey for website authentication (how it works and how to use it). Then he follows that up with “The Purism Librem Key” and how that specific USB hardware key compares to others on the market (like the YubiKey).

Once you’ve decided on a password manager, started using a set of security guidelines and even begun utilizing a hardware key, you’re probably feeling like your computers are pretty gosh-darned secure. Right?

But what if you want *more*. What if you want to be confident that not even the BIOS of your computer has been tampered with in any way?

Enter Kyle Rankin (Tech Editor for *Linux Journal*) and his article, “Tamper-Evident Boot with Heads”. Kyle breaks down how Heads is set up and how it can be used to verify, at boot, that your BIOS and kernel haven’t been messed with by dastardly villains (like the ones in *Spaceballs*).

All of these tools are powerful ways to secure your Linux systems—whether for work or personal use. But, here’s the key, they’re effective only when *used*.

In other words, no more sticky notes. ■

Send comments or feedback
via <https://www.linuxjournal.com/contact>
or email ljeditor@linuxjournal.com.

Thanks to Sponsor
PULSEWAY
for Supporting *Linux Journal*



**System Management
at Your Fingertips.**

www.pulseway.com

Want to see your company's logo here?
Find out more, <https://www.linuxjournal.com/sponsors>.

A Line in the Sand

There's a new side to choose. It helps that each of us is already on it.

By Doc Searls

Linux Journal was born in one fight and grew through a series of others.

Our first fight was for freedom. That began in 1993, when Phil Hughes started work toward a **free software** magazine. The fight for free software was still there when that magazine was born as *Linux Journal* in April 1994. Then a second fight began. That one was against all forms of closed and proprietary software, including the commercial UNIX variants that Linux would eventually defeat. We got in the fight for open source starting in 1998. (In 2005, I got a ribbon for my own small part in that battle.) And last year, we began our fight against what **Shoshana Zuboff** calls **surveillance capitalism**, and **Brett Frischmann** and **Evan Selinger** call **re-engineering humanity**.

This new fight is against actual and wannabe corporate and



Doc Searls is a veteran journalist, author and part-time academic who spent more than two decades elsewhere on the *Linux Journal* masthead before becoming Editor in Chief when the magazine was reborn in January 2018. His two books are *The Cluetrain Manifesto*, which he co-wrote for Basic Books in 2000 and updated in 2010, and *The Intention Economy: When Customers Take Charge*, which he wrote for Harvard Business Review Press in 2012. On the academic front, Doc runs ProjectVRM, hosted at Harvard's Berkman Klein Center for Internet and Society, where he served as a fellow from 2006–2010. He was also a visiting scholar at NYU's graduate school of journalism from 2012–2014, and he has been a fellow at UC Santa Barbara's Center for Information Technology and Society since 2006, studying the internet as a form of infrastructure.



FROM THE EDITOR

government overlords, all hell-bent on maintaining the caste system that reduces each of us to mere “consumers” and “data subjects” in a world [Richard Brautigan](#) described perfectly half a century ago in his poem [“All Watched Over By Machines of Loving Grace”](#). You know, like *The Matrix*, only for real.

They’ll fail, because no machine can fully understand human beings. Each of us is too different, too original, too wacky, too self-educated, too built for gaming every system meant to control us. (Discredit where due: we also suck in lots of ways. For example, [Scott Adams](#) is right that [we’re easy to hack with a good con.](#))

But why wait for nature to take its course when surveillance capitalists are busy setting civilization back decades or more—especially when we can obsolesce their whole business in the short term?

Here at *Linux Journal*, we’re already doing our part by [not participating](#) in the surveillance business that digital advertising has mostly become, and by [doing pioneering work in helping the online publishing business](#) obey the wishes of its readers.

At a deeper level, what we’ve been trying to do all along—just by working on Linux—is prove that **free people are worth more than captive ones**, both to themselves and to everyone and everything else. In *Matrix* terms, we want to [make each of us a Neo](#).

What’s hard about this isn’t making the software and hardware we need. That’s low-hanging fruit, even if most VCs haven’t seen that yet. So is getting publicity for it.

The hard thing is that lots of our friends are working to improve our lives as captives.

It doesn’t help that much—or most—of the work they do is good and necessary. We need people pushing the *status quo* in a helpful direction. There are also countless ways to improve and reform all our standing institutions, from politics to health care to education to social media and its platforms. And doing that work tends to pay, through credentials, experience and money.

And it’s so much easier to see what’s wrong in the world as it is, and to fight for changing it, than it is to see first causes at a deeper level, and work to change damn near everything

FROM THE EDITOR

with a few good lower-level hacks. (That's what we did with all the free and open-source protocols and code bases on which our networked world now utterly depends.)

So it's really hard to draw a line here: one between what we want for people as independent agents of themselves and all the ways people can work to move the *status quo* for both institutions and the people who depend on them.

But let's draw one anyway. Maybe it will help.

On one side is work toward proving free people are worth more than captive ones. On the other side is work toward making life easier for people who remain captive in existing systems.

We can sort this out with a two-part question:

1. *Are you working to give people their own ways of dealing with companies and other organizations in the world, at scale?* (By **scale**, I mean having one way to deal with many others, such as we already get with protocols like TCP/IP, HTTP, IMAP, SMTP and FTP, and with apps, such as browsers and email clients.) Or...
2. *Are you working to help companies and other organizations (for example, governments) treat people better than they do already?*

I think it's safe to say that most *Linux Journal* readers are on the first side. It's also safe to say that most people working to make life better for other people are on the other side.

So let's not try to recruit over there on the other side. Let's look instead at how best to recruit from our own ranks.

Three challenges there.

One is that everybody qualified to join and help a cause is already busy.

Another is that everybody doing good and original person-liberating work in the world is damn good at being self-liberated in the world as it is. For example, the wizards among us

FROM THE EDITOR

are very good at securing their own borders in the networked world, and at maintaining as many different login/password combinations as there are sites and services to log in to—while relatively few are working to obsolesce the whole login/password convention.

The third is that lots of us have the attitude Chris Hill lampooned in his brilliant [Switch to Linux](#) video back in 2003 (and I wrote about [here](#)). That video stars an alpha geek who says, “Linux runs on anything”, then adds, “You’ve got to config it...write some shell scripts...update your RPMs...partition your drives...patch your kernel...compile your binaries...check your version dependencies...probably do that once or twice....It’s just so easy, and so simple. I don’t know why everyone doesn’t run Linux.” That litany has changed a lot in the last 16 years, but to assume that muggles should do what wizards do is just as wrong as it ever was.

So what’s right? Glad you asked.

Last April, in [“How Wizards and Muggles Break Free from the Matrix”](#), I put up a punch list of 13 different things already being done to help break everyone free of institutions that would rather hold them captive—and to build bases for far better institutions in the process.

At the time I wrote that, I assumed that the [GDPR](#) would clear paths for work already moving forward within all 13 items on that muggle-liberating punch list. Alas, the GDPR’s single positive achievement so far has been shaking things up. That’s it. The worst thing the GDPR has done is encourage surveillance capitalists to keep doing the same damn things, only now with the “consent” of “data subjects” clicking “agree” to misleading cookie notices everywhere.

But the work proceeds, and all of it can use your help.

So please, let us know which side of the line you stand on and what you’re ready to do about it (or, better yet, already doing). Thanks. ■

Send comments or feedback
via <https://www.linuxjournal.com/contact>
or email jeditor@linuxjournal.com.

LETTERS

Why I've Been Reading LJ Since 1996

The January 2019 issue of *LJ* came to me while I was in my Summer Holidays. Since 1996, I've been an *LJ* subscriber—maybe I'm even the oldest and farthest (both conditions together) subscriber. Many times I've asked myself why I'm so loyal to the magazine. Well, the January 2019 issue came with an answer.

First was [Bryan Lunduke and his comments on the issue](#). His story about the first Linux installation made me remember my own story. I was in the second year of a PhD course on Astrophysics at Universidad de Buenos Aires (Argentina). At that time (early 1990s), all we had were PCs with Windows for personal use. The Institute had a VAX mostly used for atomic physics calculations, and it bought a SUN for Astronomical data analysis (using IRAF and AIPS) and an IBM for number-crunching (the same atomic physics). These two worked with UNIX. This was my first introduction to the UNIX world.

A young researcher came back to the Institute from the US carrying all his experience with the Internet and UNIX world. So, after some discussion, we concluded that we needed a change of paradigm. The PCs should also work with UNIX, and I helped, with my modest knowledge of UNIX, to make the move. We discovered that the Institute bought a commercial UNIX distribution for PCs that never was installed. The distro came in 5 1/4 floppy diskettes! We installed the basic command-line tools, but the X system never worked. I remember long afternoons trying to find the magic numbers to configure it. We even tried to discover some of them using an oscilloscope. It never worked.

Another young scientist, also from the US, told us about a “UNIX-like” (later we learned its name was Linux) operating system. We decided to give it a try. It was on a bunch (~30) of 3.5” diskettes. The first try we failed; the booting diskette did not pass some of the tests. But then somebody came with another one, and this time everything went smoothly. The best part was the X system. We chose an imitation of the Solaris windows manager, and it worked out of the box. We were completely euphoric. What never worked even after one week of hard work with a commercial system, took a couple hours with this free system. It was love at first sight. I will never forget that SLS distribution with kernel 0.99 PL 12. I never abandoned Linux in my life. Even when my children had their own PCs, I emphatically recommended that they install Linux. Now,

there is only one (out of six) double-boot PC at home; the rest are pure Linux.

Then, in the same *LJ* issue, [Doc Searls' gave his views about our non-gravity non-geographical internet world](#). He tells us that in his first visit to LA, he went to Mount Wilson. I had the opportunity to visit Pasadena in July 2018 during a COSPAR meeting. With three students, I rented a car and drove to Mount Wilson and had a fascinating and charming visit to the old facilities there: the 100" reflector, where Hubble discovered the expansion of the universe, and the 150' heliostat with which Hale demonstrated that sunspots are magnetic structures in essence. As a solar physicist, visiting this old monster of the solar field with my graduate students was very stimulating.

So, that's it. Linux is a community. It's my community, where I find people with similar life stories, and so besides computing, I can share other interests. And *Linux Journal* is one (the best?) way for the community to be linked. Thank you for the good job. You'll have my support if you keep going this way.

—Guigue

Doc Searls replies: Thanks, Guillermo. Everything we do here is about satisfying, keeping and growing our community. And it really helps when we find, in essays such as the one I wrote about Mt. Wilson, that a non-Linux topic strikes a responsive chord.

Re: “Put Down the Pipe”

Some of the sample shell commands in Kyle Rankin’s [“Put Down the Pipe”](#) (in the January 2019 issue) are incorrect:

```
cat file | grep "foo"  
...  
grep "foo" file
```

A more exact replacement would be:

```
grep "foo" < file
```

LETTERS

which, like the `cat ...` command, causes `grep` to read from its standard input. They're nearly equivalent, because `grep` takes a filename as an argument, but it's good to know the more general solution for commands that behave differently.

Also:

```
sort < file1 file2 | uniq
```

This won't work. The `<` redirection operator cannot take two filenames as an argument. The command is actually valid, but `file2` will be passed as an argument to `sort`. It's equivalent to this:

```
sort file2 < file1 | uniq
```

This command:

```
find ./ -name "*.mp3" -type f -print0 | rm -f
```

also doesn't work. The `find` command prints a list of filenames to `stdout` (separated by null characters due to the `-print0` argument). The `rm` command does *not* take a list of filenames from standard input. You'd have to use `xargs -0`.

And then there's this:

```
find ./ -name "*.mp3" -type f -print0 | echo
```

Similarly, the `echo` command does not read from `stdin`; you'd have to use `xargs -0`. (Incidentally, I'd use just `"."` rather than `"./"`. Adding a `/` to ensure that you're referring to a directory can be useful, but `"."` is always a directory.) Then:

```
find ./ -name "*.mp3" -type f -print0 | xargs echo  
find ./ -name "*.mp3" -type f -print0 | xargs rm -f
```

You need to use `xargs -0`.

—Keith Thompson

“Way, Way Outside!”

Marcel Gagné’s article titled “Linux and the Multiverse”, published in the January 2019 issue, featured a subsection titled “Way, Way Outside!” about PonyOS. It is always an honor to see my creations featured in media publications, especially in magazines. As I say on my website, and as Marcel reiterates in the article (though perhaps with a tone that may not convey sincerity), PonyOS is not a Linux distribution—its kernel and core applications instead are derived from my real hobby OS project ([ToaruOS](#)), where they were written from scratch by myself and a handful of contributors over nearly a decade. PonyOS is, of course, a joke—a special release, updated once a year on an equally special day. As it’s been several months since the last April Fools, much has happened in the upstream project, and as such, the latest release of PonyOS is a sort of time capsule, capturing a bygone era for both projects—rather nostalgic to me! PonyOS and its progenitor ToaruOS are not alone in the hobby OS world—there’s plenty of other great projects worth checking out like [Sortix](#), [Redox](#) and [Kolibri](#)—all with different goals and histories. Thanks again for the article, and I hope you’re looking forward to the release of PonyOS 6 in a few months—there’s a lot in store for it!

PS. There is a typo in the article: “Mimix” should be “Minix”, the OS famed for being the one Linus himself was running while writing Linux.

—K. Lange

Marcel Gagné replies: First and foremost, while I poke fun, I assure you that my intentions were sincere. I have the utmost respect for anyone who puts in as much work as went in to PonyOS, even if it’s done for laughs. For those laughs, I thank you, again sincerely. *Linux Journal* is about Linux, but it’s also about the nearly infinite world of open source where I see Linux as a kind of poster child. When planning to write about Linux distributions, I chose to wrap up my exploration by stepping outside the mainstream Linux box to other open-source OS projects, which is how PonyOS came to be featured. Thanks

for the heads up on Sortix, Redox and Kolibri. I definitely will check them out, but I think I'll start with your own project, ToaruOS. I also will keep my eyes open for PonyOS 6.

As for the “Mimix/Minix” typo, I blame that one on Discord. (My kid made me watch tons of *My Little Pony: Friendship is Magic*.)

Addendum to Zack Brown's “Non-Child Process Exit Notification Support”

Thanks for [writing about this work!](#) I just want to add that the project is ongoing, and that I plan to refresh my non-child wait work after Christian Brauner's pidfd_kill patches land. My current thinking is that a system call returning an exit handle might be a viable alternative to a new readdir-visible proc file.

One unresolved difficulty is figuring out who should be able to read a process's exit status, as in the thread [here](#). Do just parents have access? All processes, as apparently in FreeBSD? Same user only? Root?

Still, the general idea is that you should be able, somehow, to get a file descriptor from which you read(2) a siginfo_t containing exit status (like for waitid(2)), and I'm looking forward to adding this capability to Linux one way or another.

—Daniel Colascione

Re: “All Your Accounts Are Belong to Us”

Although Shawn Powers' article [“All Your Accounts Are Belong to Us”](#) was written in 2017, I think it is still very valid.

I just wanted to thank you for the article and at the same time recommend “KeepassXC”. It is an open-source fork of the well known Keepass/KeepassX.

Once again, thanks for the article.

—Guillermo Vazquez

Shawn Powers replies: Wow, I'd forgotten all about that article! The stars aligned, and your email came just in time for the Security issue, wherein I cover the topic of password managers in depth. KeepassXC is indeed one of the managers I highlight, and if I weren't so entrenched in the LastPass world (which, looking back, was the case in 2017's article too), it might be the option I'd choose. Nevertheless, thank you for the great suggestion and the perfect timing!

From Social Media

In response to “**If Your Privacy Is in the Hands of Others Alone, You Don’t Have Any**” by Doc Searls

Christine Hall @BrideOfLinux: Being an old-fashioned human being takes a lot of work in a world where most people are willing cyborgs.

Not very bright and things just got out of hand. @dluippold: Interesting ideas, but a quick note. A person can't be a controller of her own data, because you can't process your own data, and you need to identify the controller/processor to appropriate liability for violations. Great read though, I'm going to share with my team.

In response to “**The State of Desktop Linux 2019**” by Bryan Lunduke

Chan Lai Sun: I am a Linux user for 20 years and I can say that the it's best PC OS. It's FOC, and it's preloaded with much useful software. It doesn't need anti-virus software, and the OS update has never caused interruptions to PC operations.

Sameer Verma: 2019—the year of the Desktop!

SEND LJ A LETTER We'd love to hear your feedback on the magazine and specific articles. Please write us [here](#) or send email to ljeditor@linuxjournal.com.

PHOTOS Send your Linux-related photos to ljeditor@linuxjournal.com, and we'll publish the best ones here.

Some (Linux) Bugs Have All the Fun

Bugs happen.

Every minute of every hour of every day, software bugs are hard at work, biting computer users in the proverbial posterior. Many of them go unnoticed (the bugs, not the posteriors). More still rise to the illustrious level of “bugs that are minor annoyances”.

Yet sometimes, when the stars align just so, a bug manifests itself in a truly glorious way. And when I say “glorious”, I mean “utterly destructive and soul-obliterating”. Nowhere are these bugs more insidious than when they are within the operating systems (and key components) themselves.

Case in point: an October 2018 bug in an update for Windows 10 caused entire user folders to be deleted. Documents? Gone. Pictures? Like they never existed at all. This was a singular OS update that vaporized files from low-Earth orbit.

After that bug impacted roughly 1,500 Windows 10 users—before it even hit widespread distribution—Microsoft pulled the update entirely.

Then, after the engineering team in Redmond thoroughly tested and fixed this gnarly bug, they did the only obvious thing: re-release the system update—with another file-destroying issue. This time it was in their un-zip functionality. More files lost to the sands of time.

Seriously. That actually happened.

Things aren't necessarily that much better over in Apple land, either.

A little more than a year ago—at the end of November 2017—a bug occurred in Mac OS X (yeah, I know they've renamed it “macOS”, but I'm stubborn and I'll call it what I want) that allowed anyone to gain root access to any Macintosh (running the latest version of the OS) by following these extremely complex steps:

1. Turn on a Macintosh.
2. Type **root** as the user name and leave the password blank.
3. Press Enter.

I know. I know. That'll be hard to remember, right?

To Apple's credit, the company did manage to release a system update rather quickly, thus minimizing the potential damage. But, just the same, I'd say that one calls for a “yikes”—possibly even an “oh, dear”.

As satisfying as it is to make fun of Microsoft and Apple—and, boy howdy, is it ever—we in the Linux (and general Free and Open-Source Software world) are not immune from highly embarrassing, crazy destructive bugs and security vulnerabilities.

What follows are two that I find rather interesting. One is a remote exploit that had serious ramifications. The other is a local security bug that, well, I find amusing.

Note: there are lots of bugs—more than likely can be cataloged—in every system on the planet. These are just the two that I picked.

For the first one, let's travel back to the year 2014—September 24th, to be precise. Taylor Swift and Meghan Trainor were dominating the radio. The Guardians of the Galaxy were busy doing their galaxy-guarding thing.

And ShellShock was unveiled to the world: a “privilege escalation” bug (or rather, a series of related bugs) in Bash that allowed commands to be executed...that should not be accessible to that shell instance. Obviously, that’s a bad thing.

Although technically not Linux-specific (it impacted multiple systems that utilize the Bash shell), Linux was (due to its popularity in internet-facing servers) the system that got the bulk of the attention.

By the next day, September 25, 2014, attacks already were occurring that took advantage of ShellShock, including botnets targeted at critical web infrastructure and the United States Department of Defense.

Thanks to the hard work of the Bash maintainers, along with those working on various Linux distributions, the bug was patched, and the patch was released within two to three days for all the major Linux systems. Apple, who also was impacted by ShellShock, managed to release fixes a few days later.

Although these sorts of issues are never fun—and don’t make anyone look good—at least we can take comfort in the fact that we (in the Linux world) patched our systems before Apple did. Gotta take pleasure in the little things in life.

This next bug ranks in as my *favorite Linux bug of all time*. (Yes, I have a favorite bug. And, yes, I agree, that’s odd.) It goes a little something like this.

Picture yourself in December 2015, sitting in front of your lovely computer, running any of a variety of major distributions.

You turn that lovely machine on and get to the Grub (Grub2, to be precise) menu. Hit backspace. Then hit backspace again. In fact, hit backspace 26 more times (28 in total), and boom—you’re entered into a rescue shell.

What can you do in said rescue shell? Well, as it turns out, just about anything you can dream up, including, but not limited to, loading a custom Linux kernel (providing

the opportunity to rootkit the main system), deleting all manner of data and even deleting Grub itself.

But, don't worry, this impacted only any version of Grub between 2009 and 2015—so, you know, six years worth of Linux distributions (including desktops, servers, mobile devices and embedded systems). Or, as I like to call it, “Just about every important, and not-so-important, computer on Earth.” No biggie.

Once again, the maintainers of the major Linux distributions were right on the case—most with fixes pushed out to their repositories within days (if not hours) of the exploit being released to the public.

If you are somewhat new to the wonderful world of Linux and, thus, didn't get to live through those fun moments in time, never fear. If I've learned anything about software, it's this: *There'll always be more bugs. And, going on odds, the ones next year will be more destructive than the last crop.*

Let's just hope they're at least as entertaining as hitting backspace 28 times.

—*Bryan Lunduke*

Astronomy Software by Any Other Name

In this article, I introduce another option available for the astronomers out there—specifically, **Cartes du Ciel**, also known as SkyChart. Similar to other larger astronomy programs, you can use SkyChart from the desktop to the observatory.

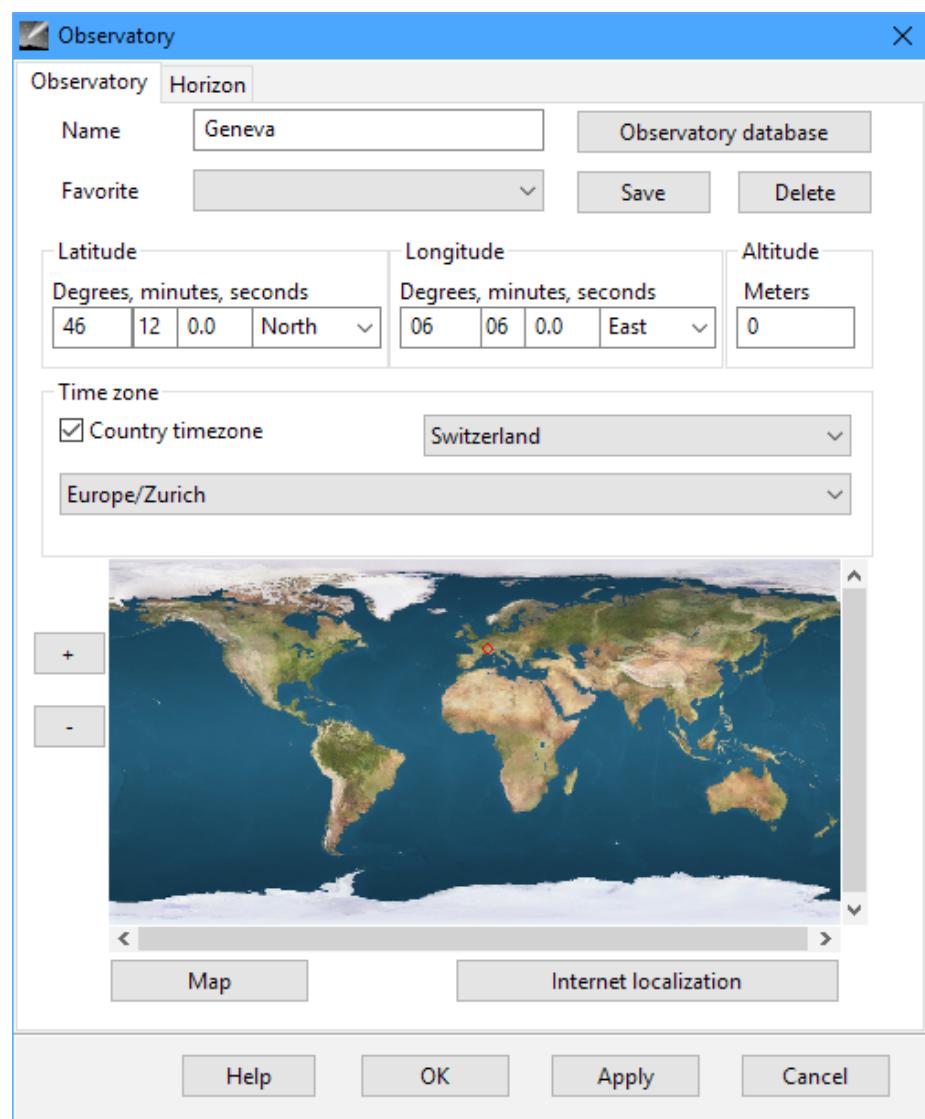


Figure 1. The first step is to set the location where you'll be making observations.

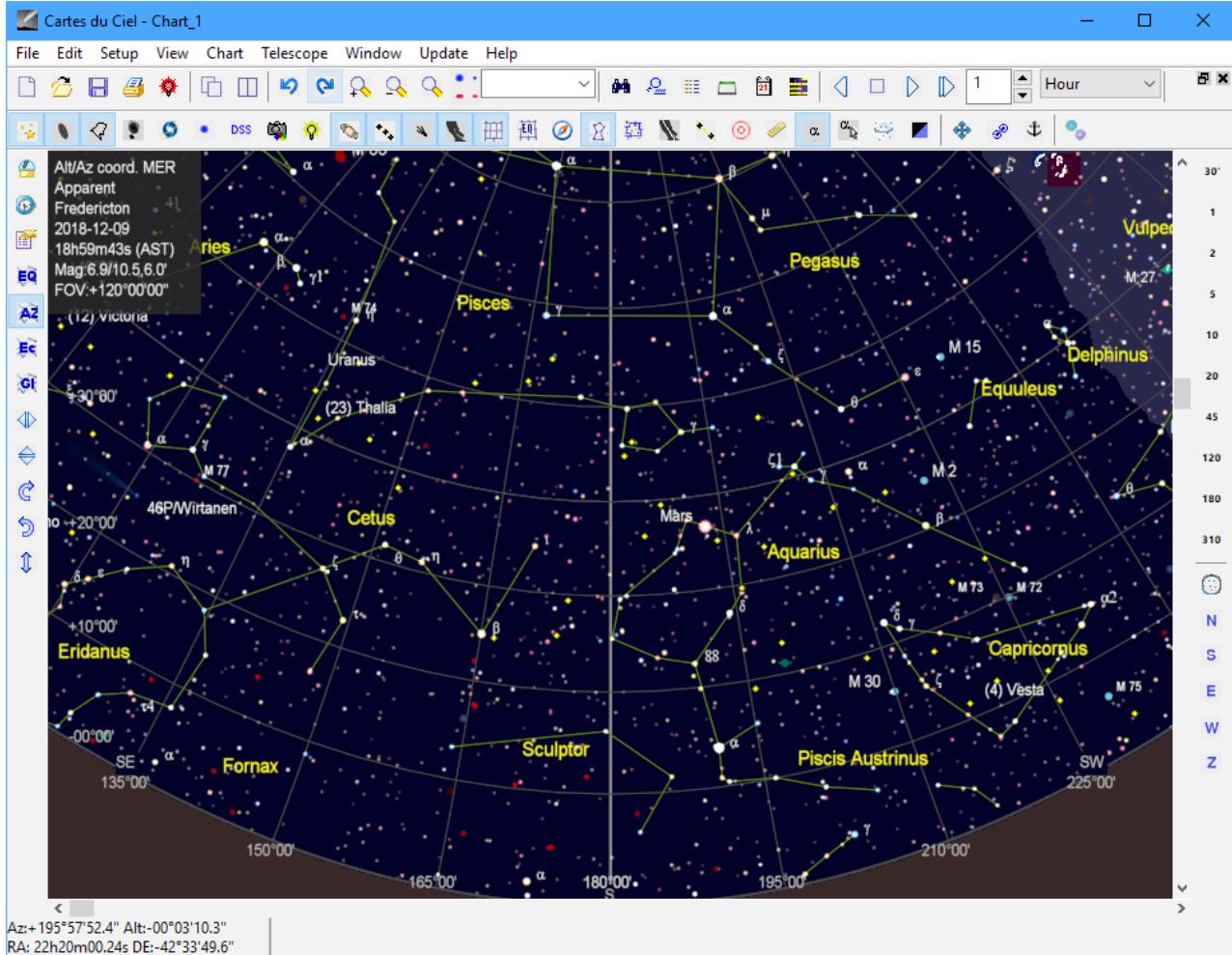


Figure 2. The initial display is the sky over your location at the current time.

SkyChart probably won't be available in your distribution's package management system, so you'll need to go to the main [website](#) to download it. DEB, RPM and TAR files are available, so you should be able to use it for just about any distribution. Downloads also are available for other operating systems and for other hardware. You even can download a version to run on a Raspberry Pi.

When you first start Cartes du Ciel, you'll be asked where on the globe your observatory is located.

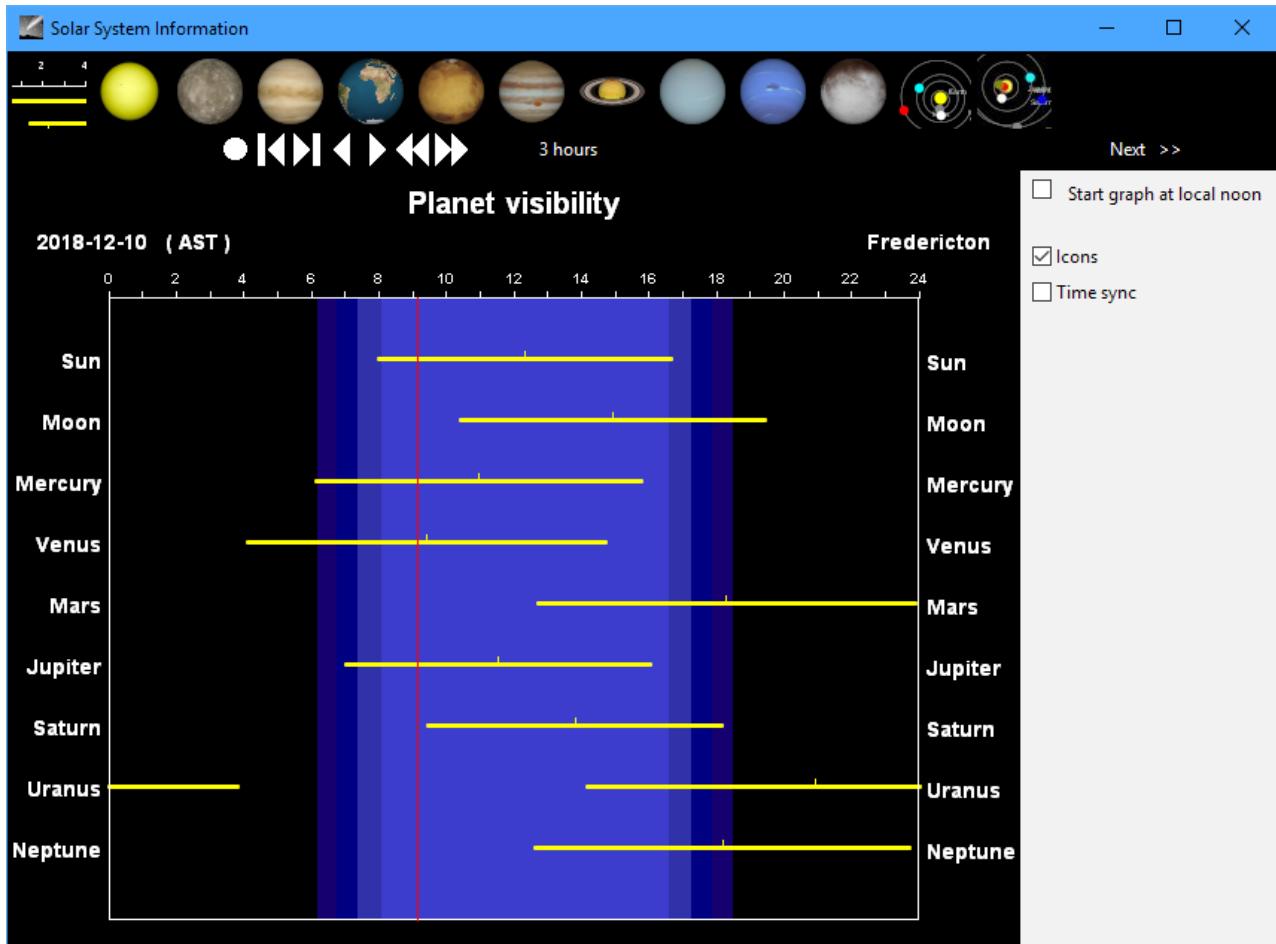


Figure 3. You can find out what each of the planets looks like from your current location for the current time.

A number of locations already are listed in the database. If your location isn't there, you can enter the latitude and longitude. Once you are done, clicking the OK button pops up a new window with the sky at the current time and location.

Unlike many other astronomy programs, time does not progress automatically. The design is more along the lines of being able to generate viewing charts for observation. Buttons in the toolbar at the top allow you to update the time easily.

The default view is to look at the sky at due south. You can change this view by

UPFRONT

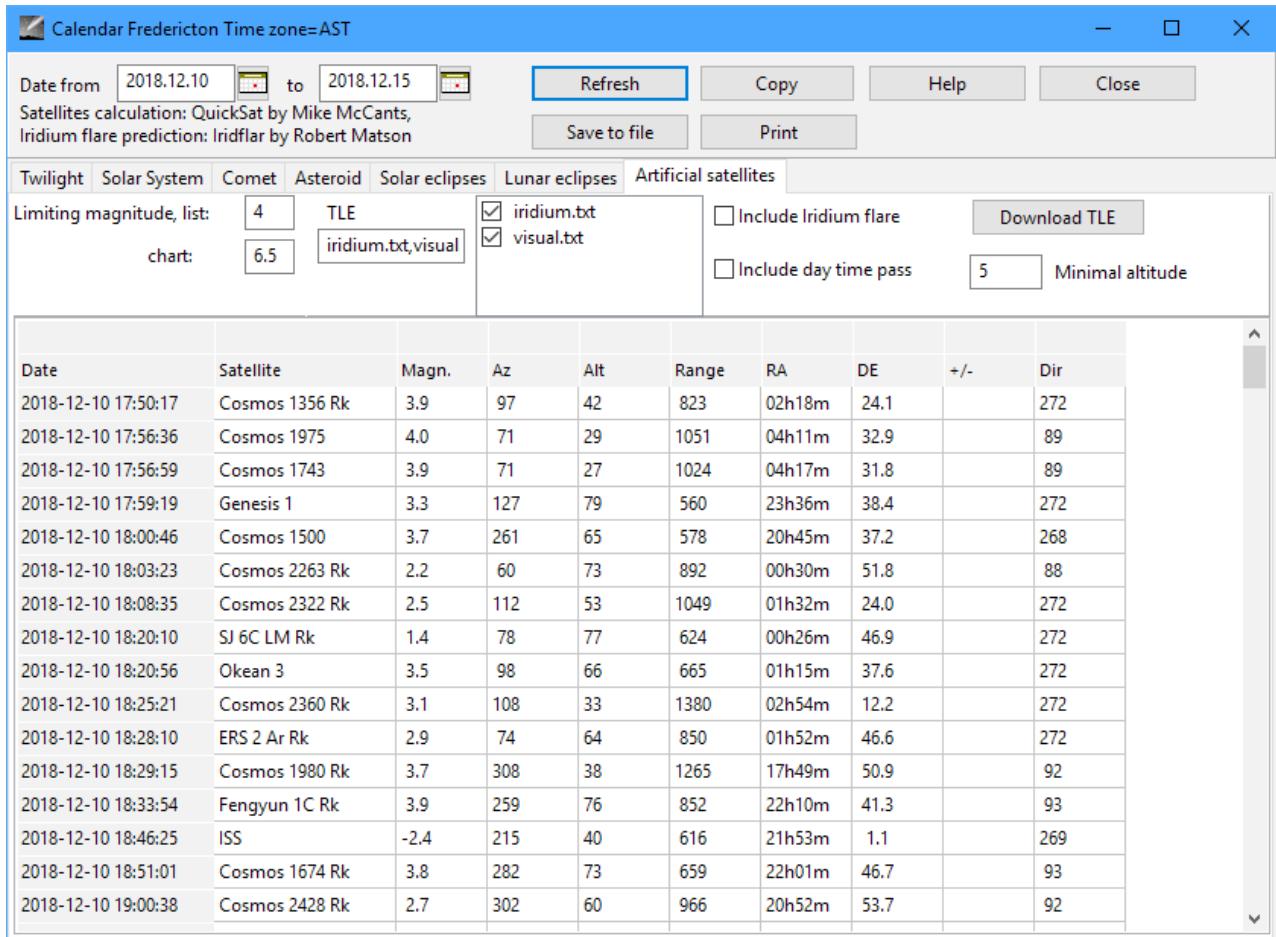


Figure 4. You can get detailed positioning and timing information for objects in the sky that can be used to point your telescope.

clicking and dragging the star field. If you want to center it on a cardinal direction, there are buttons along the bottom right-hand side of the screen for that task. Just above these cardinal direction buttons, field of view (FOV) buttons set the amount of the sky that is visible.

Along the left-hand side of the main window are several buttons for turning various coordinate systems and markers on and off. Along the top, several toolbars allow you to select which elements of the sky are visible within the sky chart that you are generating. All of these options also are available as menu items. Clicking the Chart

menu item provides a list where you can change parameters, such as the field of view, the viewing direction or the coordinate system to use.

Under the View menu item, several other informational windows are available that you can use to see even more details about what's visible in the sky. For example, when you click the menu item View→Solar System Information, you get a pop-up window showing what major objects are visible and when.

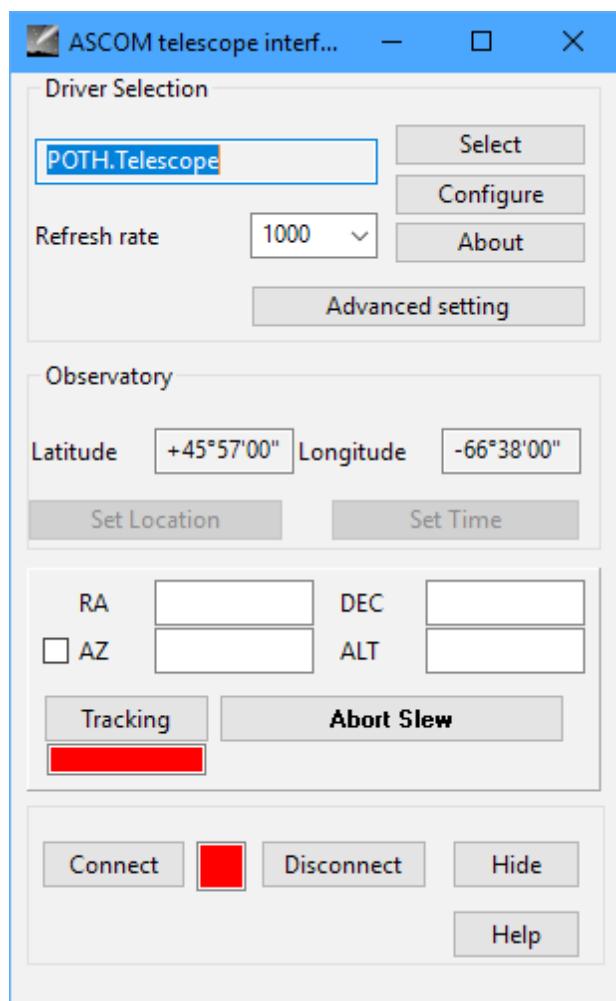


Figure 5. You can connect to a telescope that accepts computer instructions to help coordinate and direct your viewing.

Clicking each of the entries at the top of the window will pull up the relevant planet as it would look from your current location at the current time, giving a preview of what you should be able to see from your telescope. Clicking the View→Calendar menu item pops up a window with a table view of several different astronomical events for a given set of dates.

For example, you can get a list of the positions of several artificial satellites for the dates listed, or the times for twilight or the positions of comets and asteroids. You can use these types of details for ideas on where to point your telescope.

Speaking of telescopes, you also can connect directly to your telescope if it includes the functionality of being computer-controlled. Clicking the Telescope→Telescope settings menu item pops up a window where you can select from one of several telescope

control unit drivers. Once you have selected the correct driver, you can click the Telescope→Connect telescope menu item to get a new window where you can enter the parameters for your particular telescope.

Once that's done, you can get the telescope to slew or even track a given point in space. Along with being able to connect to telescopes, Cartes du Ciel also supports the SAMP (Simple Application Messaging Protocol) intercommunication protocol to talk to other pieces of equipment. The networking design is to have several units connected together through a central hub. Cartes du Ciel does not

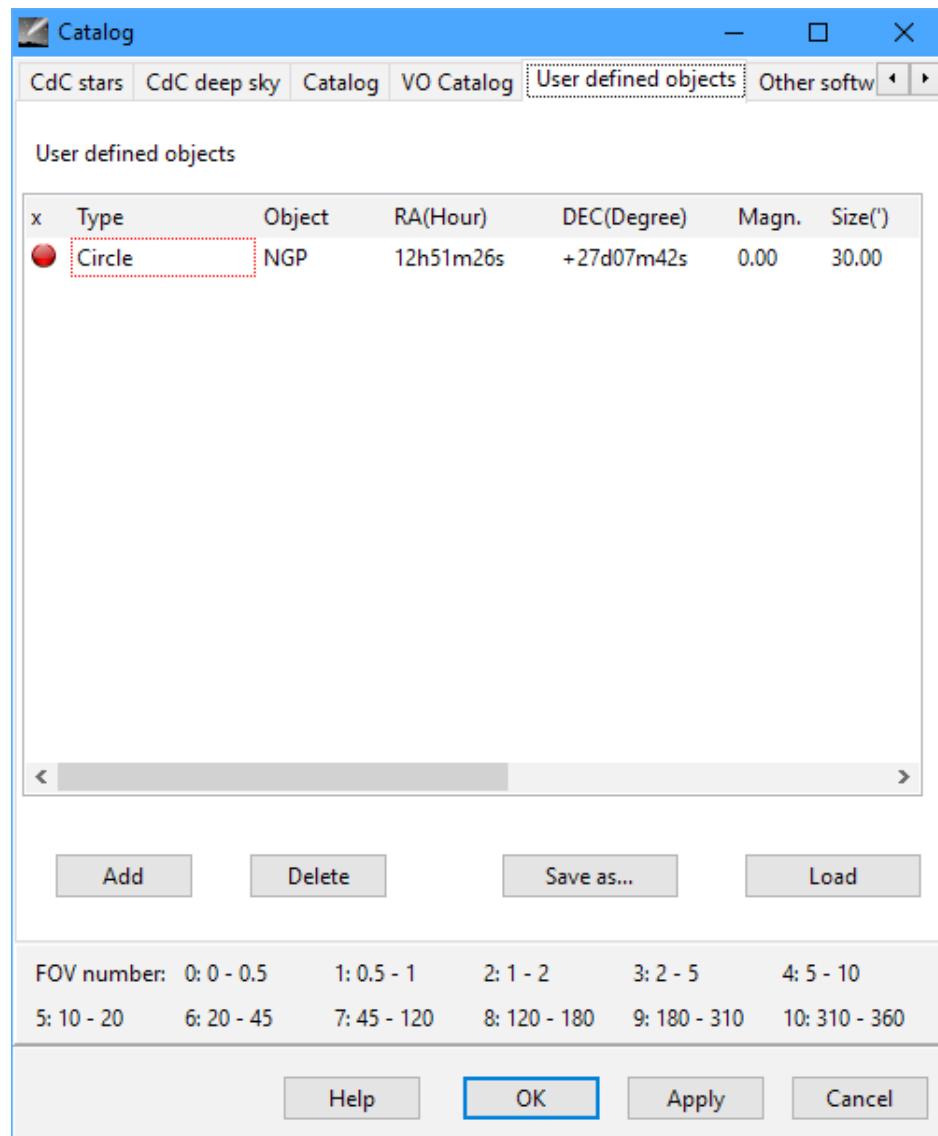


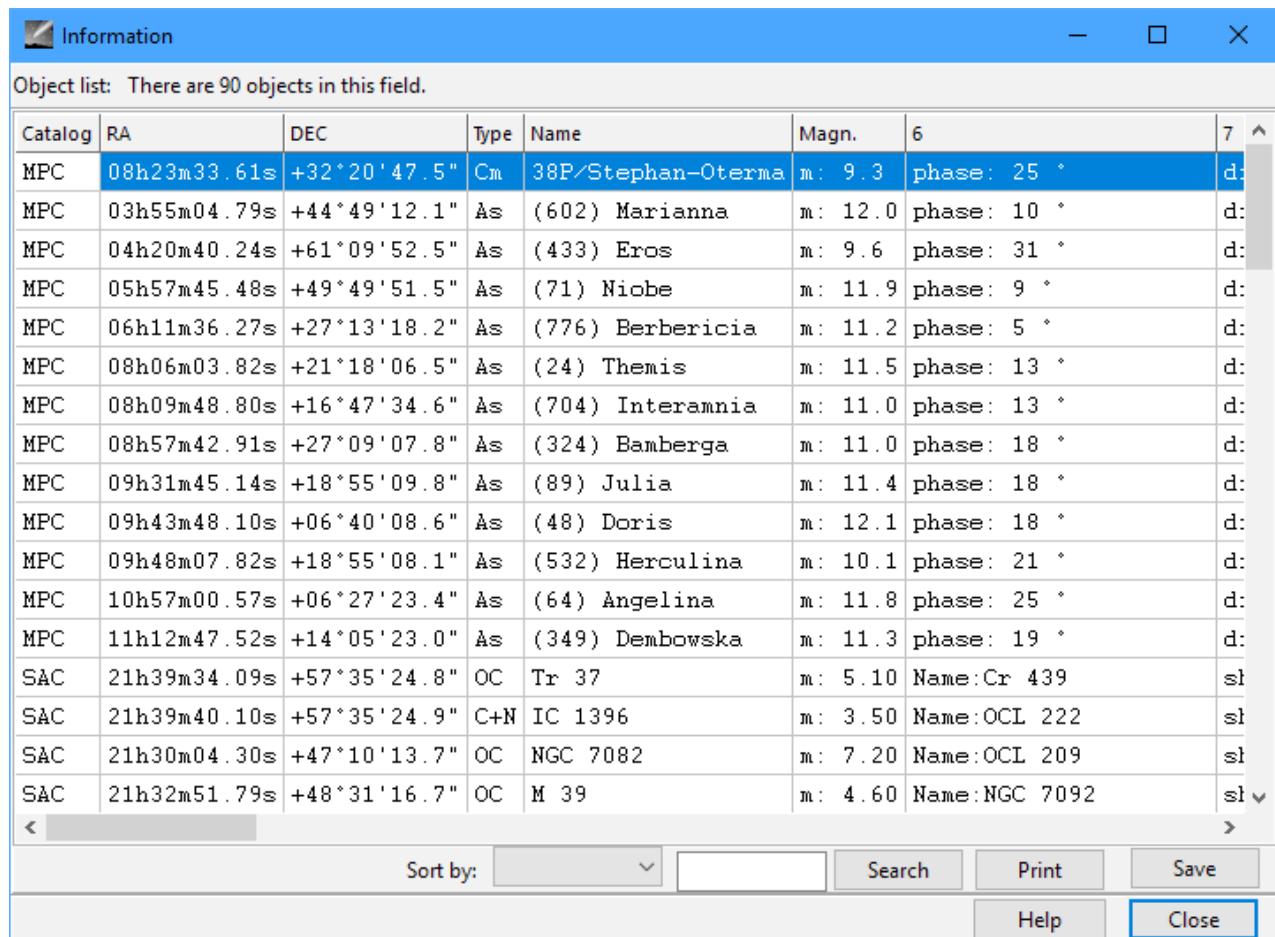
Figure 6. If necessary, you can add objects manually that may be missing from the standard catalogs of celestial objects.

UPFRONT

include software to act as a hub, so you will need some other piece of software to be the hub for your network.

Cartes du Ciel can track several categories of objects in the sky, including lots of smaller natural objects and man-made objects. These lists are being updated constantly, so functionality is included to perform updates of those catalogs. The catalogs are all located under the Update menu item.

On this list, you can update the Artificial satellites, Asteroid elements, Comet elements and the core software. Clicking on any of those menu items opens a window



The screenshot shows a window titled "Information" with a blue header bar. Below the header, a message says "Object list: There are 90 objects in this field." The main area is a table with 16 columns and 21 rows of data. The columns are labeled: Catalog, RA, DEC, Type, Name, Magn., and several others that are mostly blank or have placeholder values like "6". The first few rows show MPC objects, while the last few rows show SAC objects. The table has scroll bars on the right and bottom. At the bottom of the window are buttons for "Sort by:", "Search", "Print", "Save", "Help", and "Close".

Catalog	RA	DEC	Type	Name	Magn.	6	7	8	9	10	11	12	13	14	15
MPC	08h23m33.61s	+32°20'47.5"	Cm	38P/Stephan-Oterma	m: 9.3	phase: 25 °	d:								
MPC	03h55m04.79s	+44°49'12.1"	As	(602) Marianna	m: 12.0	phase: 10 °	d:								
MPC	04h20m40.24s	+61°09'52.5"	As	(433) Eros	m: 9.6	phase: 31 °	d:								
MPC	05h57m45.48s	+49°49'51.5"	As	(71) Niobe	m: 11.9	phase: 9 °	d:								
MPC	06h11m36.27s	+27°13'18.2"	As	(776) Berbericia	m: 11.2	phase: 5 °	d:								
MPC	08h06m03.82s	+21°18'06.5"	As	(24) Themis	m: 11.5	phase: 13 °	d:								
MPC	08h09m48.80s	+16°47'34.6"	As	(704) Interamnia	m: 11.0	phase: 13 °	d:								
MPC	08h57m42.91s	+27°09'07.8"	As	(324) Bamberga	m: 11.0	phase: 18 °	d:								
MPC	09h31m45.14s	+18°55'09.8"	As	(89) Julia	m: 11.4	phase: 18 °	d:								
MPC	09h43m48.10s	+06°40'08.6"	As	(48) Doris	m: 12.1	phase: 18 °	d:								
MPC	09h48m07.82s	+18°55'08.1"	As	(532) Herculina	m: 10.1	phase: 21 °	d:								
MPC	10h57m00.57s	+06°27'23.4"	As	(64) Angelina	m: 11.8	phase: 25 °	d:								
MPC	11h12m47.52s	+14°05'23.0"	As	(349) Dembowska	m: 11.3	phase: 19 °	d:								
SAC	21h39m34.09s	+57°35'24.8"	OC	Tr 37	m: 5.10	Name:Cr 439	sl								
SAC	21h39m40.10s	+57°35'24.9"	C+N	IC 1396	m: 3.50	Name:OCL 222	sl								
SAC	21h30m04.30s	+47°10'13.7"	OC	NGC 7082	m: 7.20	Name:OCL 209	sl								
SAC	21h32m51.79s	+48°31'16.7"	OC	M 39	m: 4.60	Name:NGC 7092	sl								

Figure 7. You can print out everything you need to be able to make your observations, even if you won't have access to any kind of computing.

that checks for any updates and automatically downloads and installs them. This way, you always can keep Cartes du Ciel as up to date as possible. But, even the most robust updating system may miss out on objects that are of special interest to you. In those cases, you can add objects to your catalog manually. Clicking the Setup→Catalog menu item pops up a new window where you can edit the details of the data catalogs available to Cartes du Ciel.

If you select the User defined objects tab, you can enter a list manually of individually defined objects so that you can add them to your sky charts.

Once you have a sky chart ready, you can output the results in a few different ways. You can save the results in a file format specific to Cartes du Ciel. You also can export the result to an image file (PNG, JPEG or BMP).

If you have to travel to get to your telescope and can't bring a computer, you may need more detail than what's available from an image. Clicking the View→Object list menu item pops up a window with the details of all of the objects in the current view of the main window. This includes everything you should need in order to direct your telescope and make appropriate observations.

I hope this short article gives you some ideas of how you can get and use the type of data that's available to professional astronomers.

—*Joey Bernard*

Patreon and *Linux Journal*

PATREON

Together with the help of *Linux Journal* supporters and subscribers, we can offer trusted reporting for the world of open-source today, tomorrow and in the future. To our subscribers, old

and new, we sincerely thank you for your continued support. In addition to magazine subscriptions, we are now receiving support from readers via Patreon on our website. LJ community members who pledge \$20 per month or more will be featured each month in the magazine. A very special thank you this month goes to:

- Appahost.com
- Chris Short
- Christel Dahlskjaer
- David Breakey
- Dr. Stuart Makowski
- Fred
- James Mayes
- Josh Simmons



BECOME A PATRON

Reality 2.0: a *Linux Journal* Podcast

Join us each week as Doc Searls and Katherine Druckman navigate the realities of the new digital world: <https://www.linuxjournal.com/podcast>.



Reality 2.0

Brought to
you by

LINUX
JOURNAL

News Briefs

Visit [LinuxJournal.com](https://www.linuxjournal.com) for daily news briefs.

- Opera announced the launch of a built-in cryptocurrency wallet for Android. According to [The Verge](#), “The wallet will first support ethereum, with support for other coins likely to come later. Ether investors using Opera would potentially be able to more easily access their tokens using the feature.” You can get Opera for Android [here](#).
- Valve’s Steam link app for Raspberry Pi 3B and 3B+ is now officially available. [Phoronix reports that](#) “This app provides similar functionality to the low-cost Steam Link dedicated device that’s been available the past few years for allowing in-home streaming of games on Steam from your personal PC(s) to living room / HTPC type setups using Steam Link.” You can get the app [here](#).
- Qt introduced [Qt for Python](#). This new offering allows “Python developers to streamline and enhance their user interfaces while utilizing Qt’s world-class professional support services”. According to the press release, “With Qt for Python, developers can quickly and easily visualize the massive amounts of data tied to their Python development projects, in addition to gaining access to Qt’s world-class professional support services and large global community.” To download Qt for Python, go [here](#).
- As of January 1, 2019, all works published in the US in 1923 will enter the public domain. The [Smithsonian reports](#) that it’s been “21 years since the last mass expiration of copyright in the U.S.” The article continues: “The release is unprecedented, and its impact on culture and creativity could be huge. We have never seen such a mass entry into the public domain in the digital age. The last one—in 1998, when 1922 slipped its copyright bond—predated Google. ‘We have shortchanged a generation,’ said Brewster Kahle, founder of the Internet Archive. ‘The 20th century is largely missing from the internet.’”
- [KStars v3.0.0](#) was released in January after four months of development. [Jasem’s](#)

[Ekosphere blog post](#) lists all the new features, including the XPlanet Solar System View developed by Robert Lancaster, significant improvements to FITS viewer GUI, scheduler improvements and more.

- Malware targeting IoT devices is growing. [BetaNews reports](#) that according to McAfee Labs, “new malware targeting IoT devices grew 72 percent with total malware growing 203 percent in the last four quarters”. The growth is partly attributed to devices being harnessed for cryptomining. See the [McAfee Labs Threats Report, December 2018](#) for all the details.
- [Mozilla announced the latest release of Firefox Focus](#), introducing enhanced privacy settings. According to the Mozilla blog, “You can choose to block all cookies on a website, no cookies at all—the default so far—third party cookies or only 3rd party tracking cookies as defined by [Disconnect’s Tracking Protection list](#). If you go with the latter option, which is new to Firefox Focus and also the new default, cross-site tracking will be prevented.” You can get the latest version of Firefox Focus from [Google Play](#) and in the [App Store](#).
- Google’s Fuchsia OS will have Android app support via Android Runtime. According to [9To5Google](#), it was expected that Fuchsia would support Android apps, and now “that suspicion has been confirmed by a new change found in the Android Open Source Project, and we can say with confidence that Fuchsia will be capable of running Android apps using the Android Runtime.” The article also notes that “How exactly Fuchsia will use the Android Runtime from there is still unclear. This includes whether the Android Runtime is able to work as expected to replace Linux kernel calls with equivalents from Fuchsia’s Zircon kernel or if ART will run inside of a Linux virtual machine using Machina, Fuchsia’s virtual machine system.”
- Linux servers equipped with poorly configured IPMI (Intelligent Platform Management Interface) cards are prone to attack. [ITPro Today](#) reports that “since November, black hat hackers have been using the cards to gain access in order to install JungleSec ransomware that encrypts data and demands a 0.3 bitcoin payment (about \$1,100 at the current rate) for the unlock key”. The post

recommends that to secure against these attacks, make sure the IPMI password isn't the default and “access control lists (ACLs) should be configured to specify the IP addresses that have access the IPMI interface, and to also configure IPMI to only listen on internal IP addresses, which would limit access to admins inside the organization’s system.”

- LinuxGizmos has published its [2019 catalog of open-spec Linux hacker boards](#). These are all “hacker-friendly, open-spec SBCs that run Linux or Android”, and LinuxGizmos provides “recently updated descriptions, specs, pricing, and links to details for all 122 SBCs.”
- [Linux 5.0-rc1](#) was released last month. Linus Torvalds wrote: “The numbering change is not indicative of anything special. If you want to have an official reason, it’s that I ran out of fingers and toes to count on, so 4.21 became 5.0. There’s no nice git object numerology this time (we’re about 6.5M objects in the git repo), and there isn’t any major particular feature that made for the release numbering either. Of course, depending on your particular interests, some people might well find a feature they like so much that they think it can do as a reason for incrementing the major number. So go wild. Make up your own reason for why it’s 5.0.”
- MIT recently released [Scratch 3](#), the latest version of its visual programming language. The [Raspberry Pi blog announced](#) it has upgraded to make this a smooth transition for those who use its [free project resources](#), “whether that be at a Code Club, CoderDojo, Raspberry Jam, or at home, so we’ve been busy upgrading our resources to work with Scratch 3”. In addition, “Scratch 3 versions of all projects in the [Code Club Scratch Modules 1–3](#) and the [CoderDojo Scratch Sushi Cards](#) are already live!” See the [post](#) for more details related to Scratch 3 on RPi.
- GitHub’s CEO Nat Friedman announced that free accounts now can create private repositories (previously only paid accounts could have private repositories). [Ars Technica reports](#) that “Now every GitHub account can create an unlimited

number of private repositories. These are still restricted—only three people can collaborate on these repositories—but a great many of those projects that once had no option but to be opened up might now be marked as private.” The *Ars Technica* article also expresses concern that one possibility with this change is that “programs that would previously have been published as open source will now be closed up forever”.

- **Bash-5.0 was released** recently. This release fixes several bugs and introduces many new features. From the release announcement: “The most notable new features are several new shell variables: BASH_ARGV0, EPOCHSECONDS, and EPOCHREALTIME. The `history` builtin can remove ranges of history entries and understands negative arguments as offsets from the end of the history list. There is an option to allow local variables to inherit the value of a variable with the same name at a preceding scope. There is a new shell option that, when enabled, causes the shell to attempt to expand associative array subscripts only once (this is an issue when they are used in arithmetic expressions). The `globasciranges` shell option is now enabled by default; it can be set to off by default at configuration time.”
- **Purism announced the fourth version of its Librem laptops.** The Librem 13 and 15 will be “now be upgraded with a 7th Gen Intel Core i7-7500U Processor with integrated HD Graphics that still works with coreboot. In addition, the Librem 15 display will be upgraded to 4K resolution. Upgraded models are available now for purchase whether you pick **Librem 13: the road warrior** or **Librem 15: the desktop replacement**.” Note that the base cost will remain the same despite these updates (the Librem 15 starts at \$1599, and the Librem 13 starts at \$1399).

Send comments or feedback
via <https://www.linuxjournal.com/contact>
or email jeditor@linuxjournal.com.

Easier Python paths with `pathlib`

A look at the benefits of using `pathlib`, the “object-oriented way of dealing with paths”.

By Reuven M. Lerner



Reuven Lerner teaches Python, data science and Git to companies around the world. You can subscribe to his free, weekly “better developers” e-mail list, and learn from his books and courses at <http://lerner.co.il>. Reuven lives with his wife and children in Modi'in, Israel.

Working with files is one of the most common things developers do. After all, you often want to read from files (to read information saved by other users, sessions or programs) or write to files (to record data for other users, sessions or programs).

Of course, files are located inside directories. Navigating through directories, finding files in those directories, and even extracting information about directories (and the files within them) might be common, but they’re often frustrating to deal with. In Python, a number of different modules and objects provide such functionality, including `os.path`, `os.stat` and `glob`.

This isn’t necessarily bad; the fact is that Python developers have used this combination of modules, methods and files for quite some time. But if you ever felt like it was a bit clunky or old-fashioned, you’re not alone.

Indeed, it turns out that for several years already, Python’s standard library has come with the `pathlib` module, which makes it easier to work with directories and files. I say “it turns out”, because although I might be a long-time developer and instructor, I discovered “`pathlib`” only in the past few months—and I must admit, I’m completely smitten.

`pathlib` has been described as an object-oriented way of dealing with paths, and this description seems quite apt to me. Rather than working with strings, instead you work with “Path” objects, which not only allows you to use all of your favorite path- and file-related functionality as methods, but it also allows you to paper over the differences between operating systems.

So in this article, I take a look at `pathlib`, comparing the ways you might have done things before to how `pathlib` allows you to do them now.

pathlib Basics

If you want to work with `pathlib`, you’ll need to load it into your Python session. You should start with:

```
import pathlib
```

Note that if you plan to use certain names from within `pathlib` on a regular basis, you’ll probably want to use `from-import`. However, I strongly recommend against saying `from pathlib import *`, which will indeed have the benefit of importing all of the module’s names into the current namespace, but it’ll also have the negative effect of importing all of the module’s names into the current namespace. In short, import only what you need.

Now that you’ve done that, you can create a new `Path` object. This allows you to represent a file or directory. You can create it with a string, just as you might do a path (or filename) in more traditional Python code:

```
p2 = pathlib.Path('..')
```

But wait a second. Do you use `pathlib.Path` to represent files or directories? The answer is “yes”. You actually can use it for both. If you’re not sure what kind of object you have, you always can ask it, with the `is_dir` and `is_file` methods:

```
p1 = pathlib.Path('hello.py')
```

```
p2 = pathlib.Path('..')
```

```
p1.is_file()
```

```
True
```

```
p2.is_file()
```

```
False
```

```
p1.is_dir()
```

```
False
```

```
p2.is_dir()
```

```
True
```

Notice that just because you create a `Path` object doesn’t mean that the file or directory actually exists. You can check that with the `exists` method:

```
>>> p1 = pathlib.Path('hello.py')
```

```
>>> p1.exists()
```

```
True
```

```
>>> p2 = pathlib.Path('asdfafafsaafaa')
```

```
>>> p2.exists()
```

```
False
```

Manipulating Paths

Let’s say you want to work with a file called `abc.txt` in the directory `/foo/bar`. In a

typical Python program, you then would say:

```
open('/foo/bar' + 'abc.txt')
```

You aren't doing anything particularly exciting here; you're just joining two strings together, the first of which represents a directory and the second of which represents a file. But as you can see, there's already a problem, in that you don't have a / separating the directory from the filename.

You can avoid such problems by using `os.path.join`:

```
>>> import os.path
>>> dirname = '/foo/bar'
>>> filename = 'abc.txt'

>>> os.path.join(dirname, filename)
'/foo/bar/abc.txt'
```

Using `os.path.join` not only ensures that there are slashes where you need them, but it also works cross-platform, using \ if your program is running on a Windows system.

That's nice, but `pathlib` offers another option: you can use the / operator, normally used for division, to join paths together. For example:

```
>>> dirname = pathlib.Path('/foo/bar')

>>> dirname / filename
PosixPath('/foo/bar/abc.txt')
```

It takes a bit of time to get used to seeing / between what you might think of as strings. But remember that `dirname` isn't a string; rather, it's a `Path` object. And / is a Python operator, which means that it can be overloaded and redefined for

different types.

If you forget and try to treat your Path object as a string, Python will remind you:

```
>>> dirname + filename  
TypeError: unsupported operand type(s) for +: 'PosixPath'  
      and 'str'
```

Working with Directories

If your Path object contains a directory, there are a bunch of directory-related methods that you can run on it. Actually, you can run these methods on non-directory Path objects as well, but it won't end very usefully or well.

For example, let's say you want to find all of the files in the current directory. You can say:

```
>>> p = pathlib.Path('.')
>>>
>>> p.iterdir()
<generator object Path.iterdir at 0x111e4b1b0>
```

Notice that the result from calling `p.iterdir()` is a generator object. You can put such an object in a `for` loop or other context that expects/requires iteration. The generator will return one value for each filename in your directory.

But, what if you're not interested in getting all of the filenames? What if you want to get only those files ending with .py? If you were working in the UNIX shell, you'd say something like `ls *.py`. Such a pattern isn't a regular expression, despite what many people believe. Rather, such a pattern is known as "globbing". The `glob` module in Python handles that for you, letting you say something like:

```
import glob
glob.glob('*.py')
```

The result of invoking `glob.glob` is a list of strings, with each string containing a filename that matches the pattern.

Path objects have similar functionality, thanks to the `glob` method. Like `iterdir`, the `glob` method returns a generator, meaning that you can use it in a `for` loop. For example:

```
>>> p.glob('*.*py')
<generator object Path.glob at 0x111b38480>

>>> for one_item in p.glob('*.*py'):
    print(f"{one_item}: {type(one_item)}")

hello.py: <class 'pathlib.PosixPath'>
reverse_lines.py: <class 'pathlib.PosixPath'>
old_test_hello.py: <class 'pathlib.PosixPath'>
```

The good news is that you get back the filenames in the directory. And the filenames already have been filtered by `glob`, so you’re getting only matches. The even better news is that you get back Path objects (in this case, `PosixPath` objects, since this example isn’t on a UNIX system), which means that you can use all the tricks you’ve enjoyed so far.

Working with Files

Once you have a file, what can you do with it? Well, one obvious candidate is to open it and read its contents. You can do that with the `read_bytes` and `read_text` methods, which return “bytes” and string objects, respectively.

Note that unlike the `read` method that you typically can run on a “file” object in Python, both `read_text` and `read_bytes` open the file, retrieve its contents and close it again. Thus, you don’t have to worry about where the internal file pointer is located or whether you’ll be reading from the start of the file or elsewhere.

However, those methods can cause problems if you read from a particularly large file. Python happily will read as much as it can into a huge string, potentially using all (or most) of the memory on your computer.

A better strategy, and a traditional one in Python, is to read through the file’s contents one line at a time. This is accomplished by putting an open “file” object into a **for** loop; file objects are iterable and return one line (that is, up to and including the following newline) in each iteration.

Note that although you certainly can use the built-in **open** function, you also can take advantage of the **open** method for **Path** objects:

```
>>> p = pathlib.Path('hello.py')

>>> for one_line in p.open():
>>>     print(one_line)
```

This will print all of the lines in the file. Notice that **open** knows how to work with a Path object just as easily as a string. However, you’ll also notice that when you print the file, the lines are double-spaced. That’s because each iteration includes the newline character, and **print** also inserts a newline character after each line it prints. You can adjust this by passing an empty string to the **end** parameter in the **print** function:

```
>>> for one_line in p.open():
>>>     print(one_line, end='')
```

Aside from opening files, you also can invoke a number of other methods on a Path object. For example, I mentioned before that you might not want to read the entirety of a large file into memory. You can check the file’s size, as well as many other attributes, using the **stat** method. This method, like the traditional **os.stat** Python function, returns a file’s size in bytes:

```
>>> p.stat().st_size  
123
```

You similarly can retrieve other items that `stat` reports, including the file's most recent modification timestamp, and IDs of the user and group that own the file.

If you want to manipulate the filename, you can do so with methods, such as `suffix`:

```
>>> p.suffix()  
'.py'
```

Conclusion

If you work with files on a regular basis from within Python programs, I suggest you look at `pathlib`. It's not revolutionary, but it does help to bring a lot of file-manipulating code under one roof. Moreover, the `/` syntax, although odd-looking at the start, emphasizes the fact that you're dealing with `Path` objects, rather than strings. And besides, it's just convenient to have access to so much functionality without having to remember where it's located. ■

Resources

`pathlib` was first proposed (and accepted) in PEP 428, which is worth reading [here](#). It has been around since Python 3.4. If you're still using Python 2.7, a package is available on PyPI with a backport, known as `pathlib2`.

Send comments or feedback
via <https://www.linuxjournal.com/contact>
or email ljeditor@linuxjournal.com.

Writing Secure Shell Scripts

Don't expose your system with sloppy scripts!

By **Dave Taylor**

Although a Linux desktop or server is less susceptible to viruses and malware than a typical Windows device, there isn't a device on the internet that isn't eventually attacked. The culprit might be the stereotypical nerd in a bedroom testing his or her hacker chops (think Matthew Broderick in *War Games* or Angelina Jolie in *Hackers*). Then again, it might be an organized military, criminal, terrorist or other funded entity creating massive botnets or stealing millions of credit cards via a dozen redirected attack vectors.

In any case, modern systems face threats that were unimaginable in the early days of UNIX development and even in the first few years of Linux as a hobbyist reimplementation of UNIX. Ah, back in the day, the great worry was about copyrighted code, and so useful tools constantly were being re-implemented from scratch to get away from the AT&T Bell Labs licenses and so forth.

I have personal experience with this too. I rewrote the *Hunt the Wumpus* game **wumpus** from scratch for BSD 4.2 when the Berkeley crowd was trying to get away from AT&T UNIX



Dave Taylor has been hacking shell scripts on Unix and Linux systems for a really long time. He's the author of *Learning Unix for Mac OS X* and *Wicked Cool Shell Scripts*. You can find him on Twitter as @DaveTaylor, and you can reach him through his tech Q&A site [Ask Dave Taylor](#).

legal hassles. I know, that's not the greatest claim to fame, but I also managed to cobble together a few other utilities in my time too.

Evolution worked backward with the internet, however. In real life, the lawless Wild West was gradually tamed, and law-abiding citizens replaced the outlaws and thugs of the 1850s and the Gold Rush. Online, it seems that there are more, smarter and better organized digital outlaws than ever.

Which is why one of the most important steps in learning how to write shell scripts is to learn how to ensure that your scripts are secure—even if it's just your own home computer and an old PC you've converted into a Linux-based media server with Plex or similar.

Let's have a look at some of the basics.

Know the Utilities You Invoke

Here's a classic trojan horse attack: an attacker drops a script called `ls` into `/tmp`, and it simply checks to see the userid that invoked it, then hands off its entire argument sequence to the real `/bin/ls`. If it recognizes `userid = root`, it makes a copy of `/bin/sh` into `/tmp` with an innocuous name, then changes its permission to setuid root.

This is super easy to write. Here's a version off the top of my head:

```
#!/bin/sh

if [ "$USER" = "root" ] ; then
    /bin/cp /bin/sh /tmp/.secretshell
    /bin/chown root /tmp/.secretshell
    /bin/chmod 4666 root /tmp/.secretshell
fi

exec /bin/ls $*
```

I hope you understand what just happened. This simple little script has created a shell that always grants its user root access to the Linux system. Yikes. Fancier versions would remove themselves once the root shell has been created, leaving no trace of how this transpired.

Because irony should be, well, *ironic*, I demonstrate above how to avoid this danger within the little trojan horse script. Never invoke programs by just their name; make sure you include their path. A script that has `ls $HOME` is begging for trouble, so fix it with `/bin/ls $HOME` instead.

An interesting additional place this risk can appear is in your PATH. Again, imagine if your PATH is set like this:

```
PATH=".:/bin:/usr/bin:$HOME/bin:/usr/local/bin"
```

95% of the time, that's no problem, and an invocation to `ls` or `cp` or even `date` will do just what you expect by failing to be found in the first directory in your PATH and so cascading down to /bin where the legit binary is stored. But what happens if you happen to be in /tmp when you invoke the command? Without realizing it, you actually invoke the trojan version and have created that root shell again (if you were root at the time, of course).

Solution: either never have dot as a directory in your PATH (my recommendation) or have it as the *last* entry in the chain, not the first.

Don't Store Passwords in Scripts

I admit, I'm not the best at this because I have some aliases that actually push passwords into my copy/paste buffer and then invoke an `ssh` or `sftp` connection to a remote computer. It's a dumb solution because it rather inevitably means that I have a shell script—or aliases file, in this case—that has lines like this:

```
PASSWORD="froB0Z69"
```

Or like this:

```
alias synth='echo secretpw | pbcopy; sftp adt@wsynth.net'
```

Solution: just don't do this. If you must, well, then at least don't use such a ridiculously obvious (and easy to identify during a scan) variable name. But really, find an alternative utility to accomplish the job, it's not worth the security risk.

Beware of Invoking Anything the User Inputs

This is a subtle one, but there's a big security risk in a simple sequence like the following:

```
echo -n "What file do you seek? "
read name
ls -l $name
```

What happens if the user enters something malicious like this as the filename:

```
. ; /bin/rm -Rf /
```

Quite dire consequences, whether the script is being invoked as root or just a regular user. Bash has some level of protection if you quote the argument, so the earlier sequence would be protected with the change to:

```
/bin/ls -l "$name"
```

But, if it's invoked as `eval /bin/ls -l "$name"`, that doesn't apply. Oh, and there's also the infamous backticks. Imagine user input like this:

```
. ` /bin/rm -Rf `/
```

This is another risky one because the `` pair is the lazy shortcut to `$()` and invokes a subshell when it's encountered on the command line. The invocation of `ls` will be performed by just such a shell too.

To fix this danger, if you have reason to believe that your script might have malicious users, scan and scrub your input. Easy solution: error out if you encounter a character that's not alphanumeric or a small set of safe punctuation marks.

Don't Use Shell Scripts for CGI Scripts

Running a Linux web server and learning about CGI scripts? It's not only tempting to use shell scripts for basic CGI functions, it's quick and easy too. Here's a script that tells you the load on the server:

```
#!/bin/sh

echo "Content-type: text/html"; echo ""
echo "Uptime on the Server:<pre>"
uptime -a
echo "</pre>"
```

This'll work fine once you get the permissions set properly. It's not too dangerous, but what if you wanted to do something similar as a home-grown search system for your site? Again, any time you have a script that runs with input from an unknown user, you've added some major risk factors.

In this case, the solution is *don't do it*. Use a compiled program instead that can implement safe and proper security or just use a third-party search system like Google Custom Search Engine to be maximally safe.

Be Smart about Your Coding

There are lots of reasons to love programming in the Linux shell, not the least of which is that it's fast and easy to prototype. But if you're really going to create a safe computing environment, you need to focus on security as you go, not realize after the fact that you did something dumb.

Some good online resources cover these topics in more depth. Check out the

WORK THE SHELL

[Shell Style Guide on GitHub](#) to get started. Also, Apple has a [document on shell script security](#) that's also well worth a read.

Be careful out there! A little extra time making sure your scripts are safe from major known risks is time well spent. ■

Send comments or feedback
via <https://www.linuxjournal.com/contact>
or email ljeditor@linuxjournal.com.

What's New in Kernel Development

By Zack Brown

Removing Profanity from the Source Tree

Warning: this article contains profanity.

Linus Torvalds recently stepped away from kernel development temporarily in order to think about how to be less harsh with developers in certain situations. Simultaneous with his departure was a patch introducing a new **Code of Conduct** into the kernel source tree. The effects of this are beginning to be felt.

Jarkko Sakkinen recently posted a patch to change a kernel comment containing the word “fuck” to use the word “hug” instead. So the code comment, “Wirzenius wrote this portably, Torvalds fucked it up” would become “Wirzenius wrote this portably, Torvalds hugged it up”.

Steven Rostedt replied to this, saying that the code in question had changed so much that the original comment was out of date, and it should just be removed entirely. He said, “that will be an accurate change with or without CoC.”

Jonathan Corbet remarked, “I'd much rather see either



Zack Brown is a tech journalist at *Linux Journal* and *Linux Magazine*, and is a former author of the “Kernel Traffic” weekly newsletter and the “Learn Plover” stenographic typing tutorials. He first installed Slackware Linux in 1993 on his 386 with 8 megs of RAM and had his mind permanently blown by the Open Source community. He is the inventor of the *Crumble* pure strategy board game, which you can make yourself with a few pieces of cardboard. He also enjoys writing fiction, attempting animation, reforming Labanotation, designing and sewing his own clothes, learning French and spending time with friends'n'family.

deletion or a rewrite over bleeping out words that somebody might not like.” And **Jiri Kosina** agreed, saying, “turning comments into something that often doesn’t make sense to anybody at all is hardly productive.”

Sergey Senozhatsky pointed out that Linus was the author of the original self-deprecating comment. He asked, “Linus has made a comment, in his own words, about his own code. Why would anyone be offended by this?”

And **Tobin C. Harding** remarked of the original code comment, “This is my favourite comment to date in the kernel source tree. Surely there are still some people working on the kernel that do so for fun. I actually laughed out loud when I first stumbled upon this file.”

In a different thread, **Kees Cook** said he agreed with removing “fuck” from the source tree, but felt that the word “hug” was not a good replacement, since it didn’t maintain the original meaning. He said:

“This API is hugged” doesn’t make any sense to me. “This API is hecked” is better, or at least funnier (to me). “Hug this interface” similarly makes no sense, but “Heck this interface” seems better. “Don’t touch my hecking code”, “What the heck were they thinking?” etc....”hug” is odd.

He added, “Better yet, since it’s only 17 files, how about doing context-specific changes? ‘This API is terrible’, ‘Hateful interface’, ‘Don’t touch my freakin’ code’, ‘What in the world were they thinking?’ etc.?”

Geert Uytterhoeven replied to Kees, saying, “As a non-native speaker, I find both replacements [‘hug’ and ‘heck’] difficult to understand. While many of the original comments are easy to grasp for +7 year olds who were never taught English, but are exposed to modern global ways of communication.” And **Matthias Brugger** also said, “I don’t think that the word ‘fuck’ is something we have to ban from the source code, but I don’t care too much. Anyway, please don’t change it to something like heck as it might be difficult for non-English speakers to understand.”

Some developers just shook their heads in bewilderment. **Davidlohr Bueso** remarked of Jarkko's original patch, "I hope this is some kind of joke. How would anyone get offended by reading technical comments? This is all beyond me."

John Paul Adrian Glaubitz added, "We're all grown up and don't freak out when a piece of text contains the word 'fuck'. I still don't understand why people think that the word 'fuck' is what would keep certain groups from contributing to the Linux kernel. In all seriousness, it doesn't."

And **Jens Axboe** said, "Agree, this is insanity."

David Miller also said, "Whether or not it is a joke, it is censorship. And because of that, I have no intention to apply any patches like this to any code I am in charge of."

At one point Jarkko pointed to the part of the Code of Conduct he relied on when posting his original patch: "Harassment includes the use of abusive, offensive or degrading language, intimidation, stalking, harassing photography or recording, inappropriate physical contact, sexual imagery and unwelcome sexual advances or requests for sexual favors."

He felt that the word "fuck" clearly fell into the category of offensive language.

James Bottomley replied, "No, because use of what some people consider to be bad language isn't necessarily abusive, offensive or degrading. Our most heavily censored medium is TV and 'fuck' is now considered acceptable in certain contexts on most channels in the UK and EU."

Taking another tack, James also pointed out that the Documentation/process/code-of-conduct-interpretation.rst file said specifically, "contributions submitted for the kernel should use appropriate language. Content that already exists predating the Code of Conduct will not be addressed now as a violation." Which, James said, "definitely means there should be no hunting down of existing comments in kernel code."

Jarkko replied, “Ugh, was not aware that there two documents.”

The discussion petered out shortly thereafter, but this is the sort of discussion we can expect to see again and again on the linux-kernel mailing list, as long as the Code of Conduct retains its current form.

The interesting thing for me is that the original issue had to do specifically with Linus' rough statements toward developers in specific situations. If he felt that someone should know better regarding a given issue, and already had been told how a given patch or feature should be done, but still persisted in trying to get a rejected patch or feature into the kernel, Linus might yell at them.

Somehow this has morphed into removing banned sets of “unacceptable” words from code comments. And, it did this in a brief matter of a few weeks. I wonder what else is in store.

Fun Little Tidbits in a Howling Storm (Re: Intel Security Holes)

Some kernel developers recently have been trying to work around the massive, horrifying, long-term security holes that have recently been discovered in **Intel** hardware. In the course of doing so, there were some interesting comments about coding practices.

Christoph Hellwig and **Jesper Dangaard Brouer** were working on mitigating some of the giant speed sacrifices needed to avoid Intel's gaping security holes. And, Christoph said that one such patch would increase the networking throughput from 7.5 million packets per second to 9.5 million—a 25% speedup.

To do this, the patch would check the kernel's “fast path” for any instances of **dma_direct_ops** and replace them with a simple direct call.

Linus Torvalds liked the code, but he noticed that Jesper and Christoph's code sometimes would perform certain tests before testing the fast path. But if the kernel

diff -u

actually were taking the fast path, those tests would not be needed. Linus said, “you made the fast case unnecessarily slow.”

He suggested that switching the order of the tests would fix it right up. He added:

In fact, as a further micro-optimization, it might be a good idea to just specify that the `dma_is_direct()` ops is a special pointer (perhaps even just say that “NULL means it’s direct”), because that then makes the fast-case test much simpler (avoids a whole nasty constant load, and testing for NULL in particular is often much better).

But that further micro-optimization absolutely *requires* that the ops pointer test comes first. So making that ordering change is not only “better code generation for the fast case to avoid extra cache accesses”, it also allows future optimizations.

Regarding Linus’ micro-optimization, Christoph explained:

I wanted to do the NULL case, and it would be much nicer. But the arm folks went to great lengths to make sure they don’t have a default set of dma ops and require it to be explicitly set on every device to catch cases where people don’t set things up properly, and I didn’t want to piss them off....But maybe I should just go for it and see who screams, as the benefit is pretty obvious.

Linus also suggested that for Christoph’s and Jesper’s tests, the `dma_is_direct()` function should be sure to use the `likely()` call. And this was interesting because `likely()` is used to alert the compiler that a block of code is more “likely” to be run than another in order to optimize it. And, Christoph wasn’t sure this was true. He said, “Yes, for the common case, it is likely. But if you run a setup where you say always have an iommu, it is not, in fact, it is never called in that case, but we only know that at runtime.”

So Christoph was concerned about misleading the compiler and generating worse code. But Linus explained:

Note that “likely()” doesn’t have any really huge overhead—it just makes the compiler move the unlikely case out-of-line.

Compared to the overhead of the indirect branch, it’s simply not a huge deal, it’s more a mispredict and cache layout issue.

So marking something “likely()” when it isn’t doesn’t really penalize things too much. It’s not like an exception or anything like that, it’s really just a marker for better code layout.

And that was it. Helpful hints in a time of desperate sorrow. These Intel hardware security holes are almost beyond belief. And we keep hearing about new batches of them being discovered all the time, or new exploits that require different workarounds from the ones already in place.

I’m sure Intel is working like mad to address all of this in future generations of its hardware. But the thing about security holes is that they are, by definition, hard to discover. Hardware manufacturers can poke and prod their products all they please and still miss the thing that a lone actor out in the world discovers one day by mistake. This time, it was Intel; next time, it’ll be something else. Kudos to Intel for working with the OS people in spite of the public embarrassment to find good workarounds for these problems.

Disk Encryption for Low-End Hardware

Eric Biggers and **Paul Crowley** were unhappy with the disk encryption options available for **Android** on low-end phones and watches. For them, it was an ethical issue. Eric said:

We believe encryption is for everyone, not just those who can afford it. And while it’s unknown how long CPUs without AES support will be around, there will likely always be a “low end”; and in any case, it’s immensely valuable to provide a software-optimized cipher that doesn’t depend on hardware support. Lack of hardware support should not be an excuse for no encryption.

Unfortunately, they were not able to find any existing encryption algorithm that was both fast and secure, and that would work with existing Linux kernel infrastructure. They, therefore, designed the **Adiantum encryption mode**, which [they described in a light, easy-to-read and completely non-mathematical way](#).

Essentially, Adiantum is not a new form of encryption; it relies on the **ChaCha stream cipher** developed by **D. J. Bernstein** in 2008. As Eric put it, “Adiantum is a construction, not a primitive. Its security is reducible to that of XChaCha12 and AES-256, subject to a security bound; the proof is in Section 5 of our paper. Therefore, one need not ‘trust’ Adiantum; they only need trust XChaCha12 and AES-256.”

Eric reported that Adiantum offered a 20% speed improvement over his and Paul’s earlier **HPolyC encryption mode**, and it offered a very slight improvement in actual security.

Eric posted some patches, adding Adiantum to the Linux kernel’s crypto API. He remarked, “Some of these patches conflict with the new ‘Zinc’ crypto library. But I don’t know when Zinc will be merged, so for now, I’ve continued to base this patchset on the current ‘cryptodev’.”

Jason A. Donenfeld’s Zinc (“Zinc Is Not crypto”) is a front-runner to replace the existing kernel crypto API, and it’s more simple and low-level than that API, offering a less terrifying coding experience.

Jason replied to Eric’s initial announcement. He was very happy to see such a good disk encryption alternative for low-end hardware, but he asked Eric and Paul to hold off on trying to merge their patches until they could rework them to use the new Zinc security infrastructure. He said, “In fact, if you already want to build it on top of Zinc, I’m happy to work with you on that in a shared repo or similar.”

He also suggested that Eric and Paul send their paper through various academic circles to catch any unanticipated problems with their encryption system.

But Paul replied:

Unlike a new primitive whose strength can only be known through attempts at cryptanalysis, Adiantum is a construction based on well-understood and trusted primitives; it is secure if the proof accompanying it is correct. Given that (outside competitions or standardization efforts) no-one ever issues public statements that they think algorithms or proofs are good, what I'm expecting from academia is silence :) The most we could hope for would be getting the paper accepted at a conference, and we're pursuing that but there's a good chance that won't happen simply because it's not very novel. It basically takes existing ideas and applies them using a stream cipher instead of a block cipher, and a faster hashing mode; it's also a small update from HPolyC. I've had some private feedback that the proof seems correct, and that's all I'm expecting to get.

Eric also replied, regarding Zinc integration:

For now I'm hesitant to completely abandon the current approach and bet the farm on Zinc. Zinc has a large scope and various controversies that haven't yet been fully resolved to everyone's satisfaction, including unclear licenses on some of the essential assembly files. It's not appropriate to grind kernel crypto development to a halt while everyone waits for Zinc.

He added that if Zinc is ready, he'd be happy to use it. He just wasn't sure whether it was.

However, in spite of the uncertainty, Eric later said, "I started a branch based on Zinc: <https://git.kernel.org/pub/scm/linux/kernel/git/ebiggers/linux.git>, branch 'adiantum-zinc'."

He listed the work he'd done so far and the work that remained to be done. But regarding Zinc's remaining non-technical issues, he said:

Both myself and others have expressed concerns about these issues previously too,

yet they remain unaddressed nor is there a documentation file explaining things. So please understand that until it's clear that Zinc is ready, I still have to have Adiantum ready to go without Zinc, just in case.

Jason was happy to see the Zinc-based repository and promised to look it over. He also promised to add a documentation file covering many of Eric's concerns before posting another series of Zinc patches. And as far as Eric and Paul being ready to go without Zinc integration, he added, "I do really appreciate you taking the time, though, to try this out with Zinc as well. Thanks for that."

Meanwhile, **Herbert Xu** accepted Eric and Paul's original patch-set, so there may be a bit of friendly shuffling as both Zinc and Adiantum progress.

It's nice to see this sort of attention being given to low-end hardware. But, it's nothing new. The entire Linux kernel is supposed to be able to run on absolutely everything—or at least everything that's still in use in the world. I don't think there are too many actual **386 systems** in use anymore, but for real hardware in the real world, pretty much all of it should be able to run a fully featured Linux OS.

Note: if you're mentioned in this article and want to send a response, please send a message with your response text to leditor@linuxjournal.com and we'll run it in the next Letters section and post it on the website as an addendum to the original article. ■

Send comments or feedback
via <https://www.linuxjournal.com/contact>
or email leditor@linuxjournal.com.



**Decentralized
Certificate Authority
and Naming**

Free and open source contributors only:

handshake.org/signup

DEEP DIVE

SECURITY



Password Manager Roundup

If you can remember all of your passwords, they're not good passwords.

By Shawn Powers

I used to teach people how to create “good” passwords. Those passwords needed to be lengthy, hard to guess and easy to remember. There were lots of tricks to make your passwords better, and for years, that was enough.

That's not enough anymore.

It seems that another data breach happens almost daily, exposing sensitive information for millions of users, which means you need to have separate, secure passwords for each site and service you use. If you use the same password for any two sites, you're making yourself vulnerable if any single database gets compromised.

There's a much bigger conversation to be had regarding the best way to protect data. Is the “password” outdated? Should we have something better by now? Granted, there is two-factor authentication, which is a great way to help increase the security on accounts. But although passwords remain the main method for protecting accounts and data, there needs to be a better way to handle them—that's where password managers come into play.

The Best Password Manager

No, I'm not burying the lede by skipping all the reviews. As Doc Searls, Katherine

Druckman and myself discussed in [Episode 8 of the Linux Journal Podcast](#), the best password manager is the one you use. It may seem like a cheesy thing to say, but it's a powerful truth. If it's more complicated to use a password manager than it is to re-use the same set of passwords on multiple sites, many people will just choose the easy way.

Sure, some people are geeky enough to use a password manager at any cost. They understand the value of privacy, understand security, and they take their data very seriously. But for the vast majority of people, the path of least resistance is the way to go. Heck, I'm guilty of that myself in many cases. I have a Keurig coffee machine, not because the coffee is better, but because it's more convenient. If you've ever eaten a Hot Pocket instead of cooking a healthy meal, you can understand the mindset that causes people to make poor password choices. If the goal is having smart passwords, it needs to be easier to use smart passwords than to type "password123" everywhere.

The Reason It Might Work Now

Mobile devices have become the way most people do most things online. Heck, Elon Musk said that we've become cybernetic beings, it's just that the bandwidth to our cybernetic components is really slow (that is, typing on our phones). It's always been possible to have some sort of password management app on your phone, but until recently, the operating systems didn't integrate with password managers. That meant you'd have to go from one app into your password manager, look up the site/app, copy the password, switch back to the app, paste the password, and then hope you got it right. Those days are thankfully in the past.

Both recent Android systems and iOS (Apple, not Cisco) versions allow third-party password managers to integrate directly into the data entry system. That means when you're using a keyboard to type in a login or password, in any app, you can pull in a password manager and enter the data directly with no app switching. Plus, if you have biometrics enabled, most of the time you can unlock your password database with a fingerprint or a view of your face. (For those concerned about the security of biometric-only authentication, it can, of course, be turned off, but remember how

important ease of use is for most people!)

So although password managers have been around for years and years, I truly believe it's only with the advent of their integration into the main operating system of mobile devices that people will actually be able to use them widely. Not all Linux users will agree with me, and not all people in general will want their passwords available in such an easy manner. For the purpose of this article, however, a mobile option is a necessity.

A Tale of Two Concepts

Remember when “the cloud” was a buzzword that didn’t really mean anything specific, but people used it all the time anyway? Well, now it very clearly means servers or services run on computers you don’t own, in data centers you don’t control. The “cloud” is both awesome and terrible. When it comes to storing password data, many people are rightfully concerned about cloud storage. When it comes to password managers, there are basically two types: the kind that stores everything in a local database file and those that store the database in the cloud.

The cloud-based storage isn’t as unsettling as it seems. When the database is stored on the “servers in the sky”, it’s encrypted before it leaves your device. Those companies don’t have access to your actual passwords, just the highly encrypted database that holds them—as long as you trust the companies to be honest about such things. For what it’s worth, I do think the major companies are fairly trustworthy about keeping their grubby mitts off your actual passwords. Still, with the closed-source options, a level of trust is required that some people just aren’t willing to give. I’m going to look at password managers from both camps.

The Contenders

I picked five(-ish) password managers for this review. Please realize there are dozens and dozens of very usable, very secure, password managers for Linux. Some are command-line only. Some are just basic PGP encryption of text files containing user name/password pairs. Today’s review is not meant to be all-encompassing; it’s meant to be helpful for average Linux users who want to handle their passwords better than they currently do. I

say five (-ish), because one of the entries has multiple versions. The list is:

1. KeePass/KeePassX/KeePassXC: this is the one (-ish) that has multiple variations on the same theme. More details later.
2. 1Password.
3. LastPass.
4. Bitwarden.
5. Browser.

I highlight each of these in this article, in no particular order.

Your Browser's Password Database

Most people don't consider using their browser as a password manager a good idea. I'm one of those people. Depending on the browser, the version and the settings you choose, your passwords might not even be encrypted. There is also the problem of using those passwords in other apps. Granted, if you use Chrome, your Android phone likely will be able to access the passwords for you to use in other apps, but I'm simply not convinced the browser is the best place to store your passwords.

I'm sure the password storage feature of modern browsers is more secure than in the past, but a browser's main function isn't to secure your passwords, so I wouldn't trust it to do so. I mention this option because it's installed by default with every browser. It's probably the most widely used option, and that breaks my heart. It's too easy to click "save my password" and conveniently have your password filled in the next time you visit.

Is using the browser's "save password" function better than using nothing at all? Maybe. It does allow people to use different passwords, trusting the browser to remember them. But, that's about it. I'm sure the latest browsers have the option to secure the passwords a bit, but it's not that way by default. I know this, because

when I sit at my wife's computer, I simply start her browser (Chrome), and all her passwords are filled in for me when I visit various websites. They've almost made it too easy to use poor security practices. The only hope is to have better options that are even easier—and I think we actually do. Keep reading!

The KeePass Kraziness

First off, these password managers are the ones that use a local, non-cloud-based database for storing passwords. If the thought of your encrypted passwords living on someone else's servers offends your sensibilities, this is probably the best choice for you. And it is a really good choice, whichever flavor you pick.

The skinny on the various programs that share similar names is that originally, there was KeePass. It didn't have a Linux version, so there was another program, KeePassX, that used an identical (and fully compatible) database. KeePassX runs natively on Linux, along with the other major OSes. To complicate issues, KeePass then released a Linux version, which runs natively, but it uses Mono libraries. It runs, and it runs fine, but Mono is a bit kludgy on Linux, so most folks still used KeePassX. Then KeePassXC came around, because the KeePassX program was getting a little long in the tooth, and it hadn't been updated in a long time. So now, there are three programs, all of which work natively on Linux, and all of which are perfectly acceptable programs to use. I prefer KeePassXC (Figure 1), but only because it seems to be most actively developed. The good news is, all three programs can use the exact same database file. Really. If there is a single ray of sunshine on a messy situation, it's that.

KeePass(X/XC) Features:

- Local database file, with no syncing mechanism.
- Database can be synced by a third party (such as Dropbox).
- Supports master password and/or keyfile unlocking.
- Very nice password generator (Figure 2).

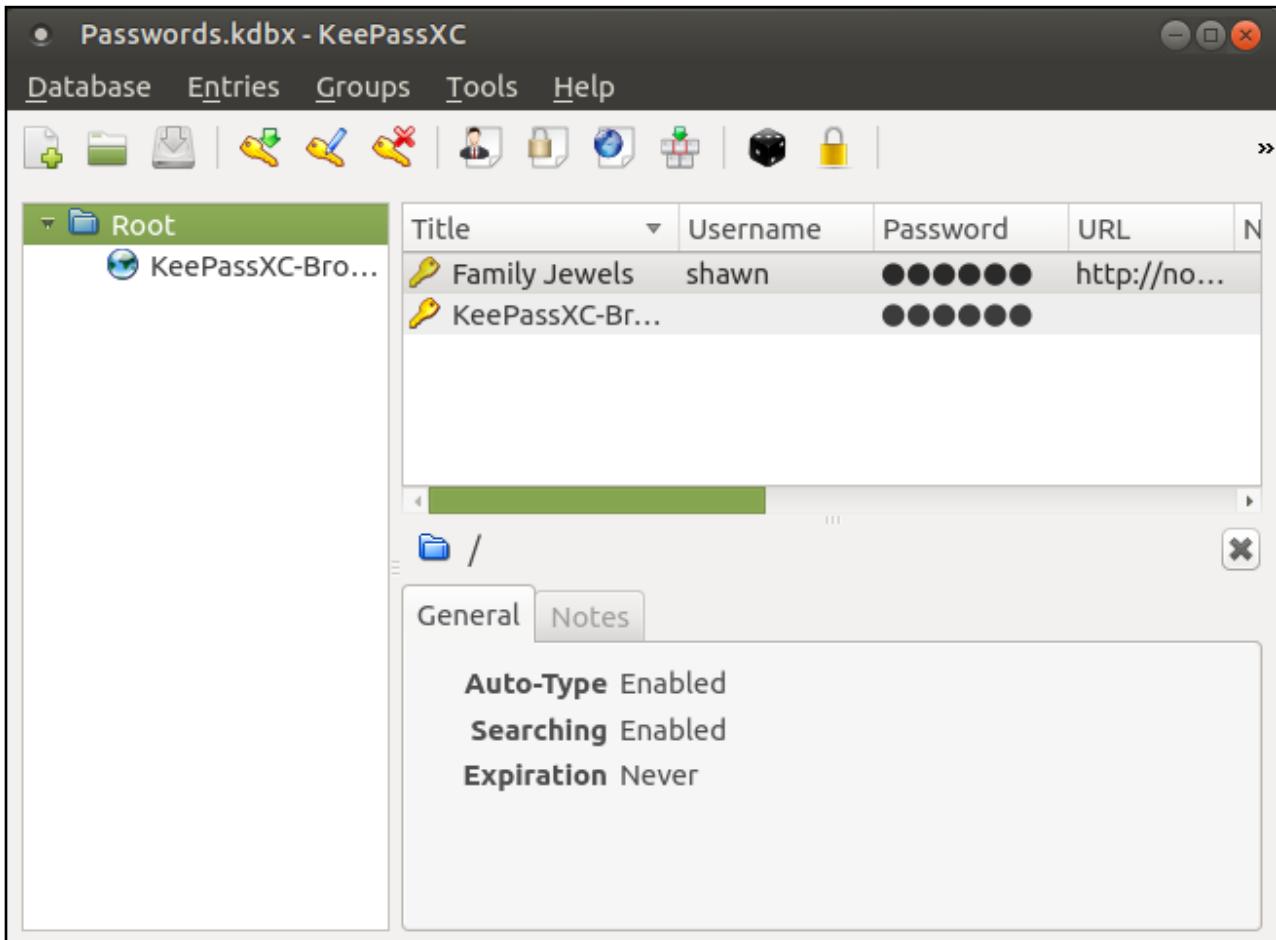


Figure 1. KeePassXC has a friendly, native Linux interface.

- Secure localhost-only browser integration (KeePassHTTP).

KeePass(X/XC) Pros:

- No cloud storage.
- Command-line interface included.
- 2FA abilities (YubiKey).

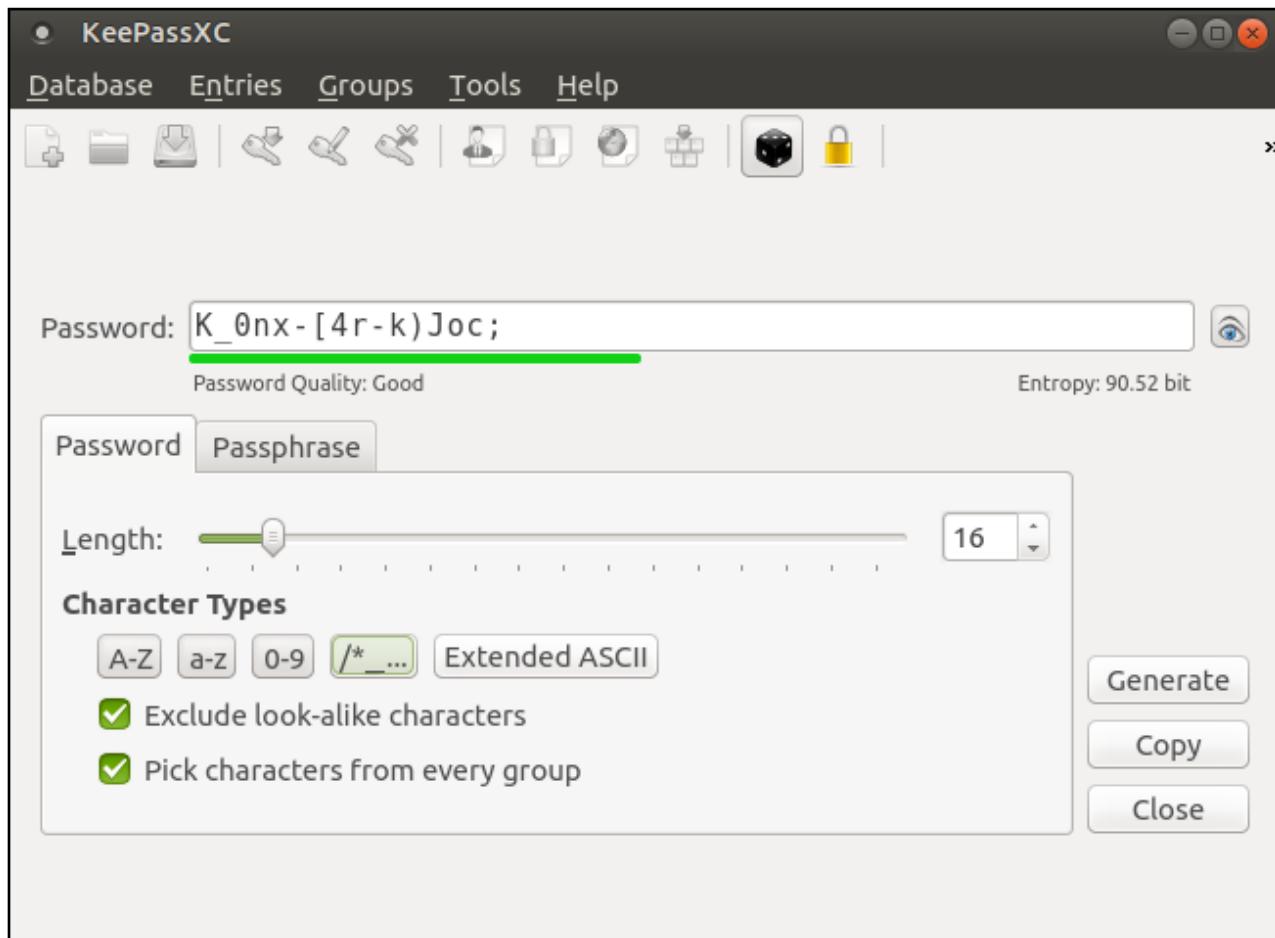


Figure 2. The KeePassXC password generator is awesome. I don't even use KeePassXC for my password manager, but I still like the generator!

- Open source.
- No “premium” features, everything is free.

KeePass(X/XC) Cons:

- No cloud storage (yes, it's a pro and a con, depending).
- Brand confusion with multiple variations.

- Requires third-party Android/iOS app for mobile use.
- More complicated than cloud-based alternatives (file to sync/copy).

The KeePass family of password managers is arguably the most open-source-minded option of those I cover here. Depending on the user, to handle syncing/copying the database rather than depending on an unknown third party to store the data has a traditional Linux feel. For those folks who are most concerned about their data integrity, a KeePass database is probably the best option. Thankfully, due to third-party tools like KeePass2Droid (for Android) and MiniKeePass/KyPass for iOS, it's possible to use your database on mobile devices as well. In fact, most apps handle syncing your database for you.

Bitwarden

I didn't know the Bitwarden password manager even existed until we did a Twitter poll asking what password managers *LJ* readers used. I have to admit, it's an impressive system, and it ticks almost all the "feel good" boxes Linux users would want (Figure 3). Not only is it open source, but also the non-premium offering is a complete system. Yes, there is a premium option for \$10/year, but the non-paid version isn't crippled in any way.

Bitwarden does store your data in its own cloud servers, but since the software is open source, you can examine the code to make sure the company isn't doing anything underhanded. Bitwarden also has its own apps for Android/iOS and extensions for all major browsers. There's no need to use a third-party tool. In fact, it even includes command-line tools for those folks who want to access the database in a text-only environment.

Bitwarden Features:

- Open-source.
- Cloud-based storage.

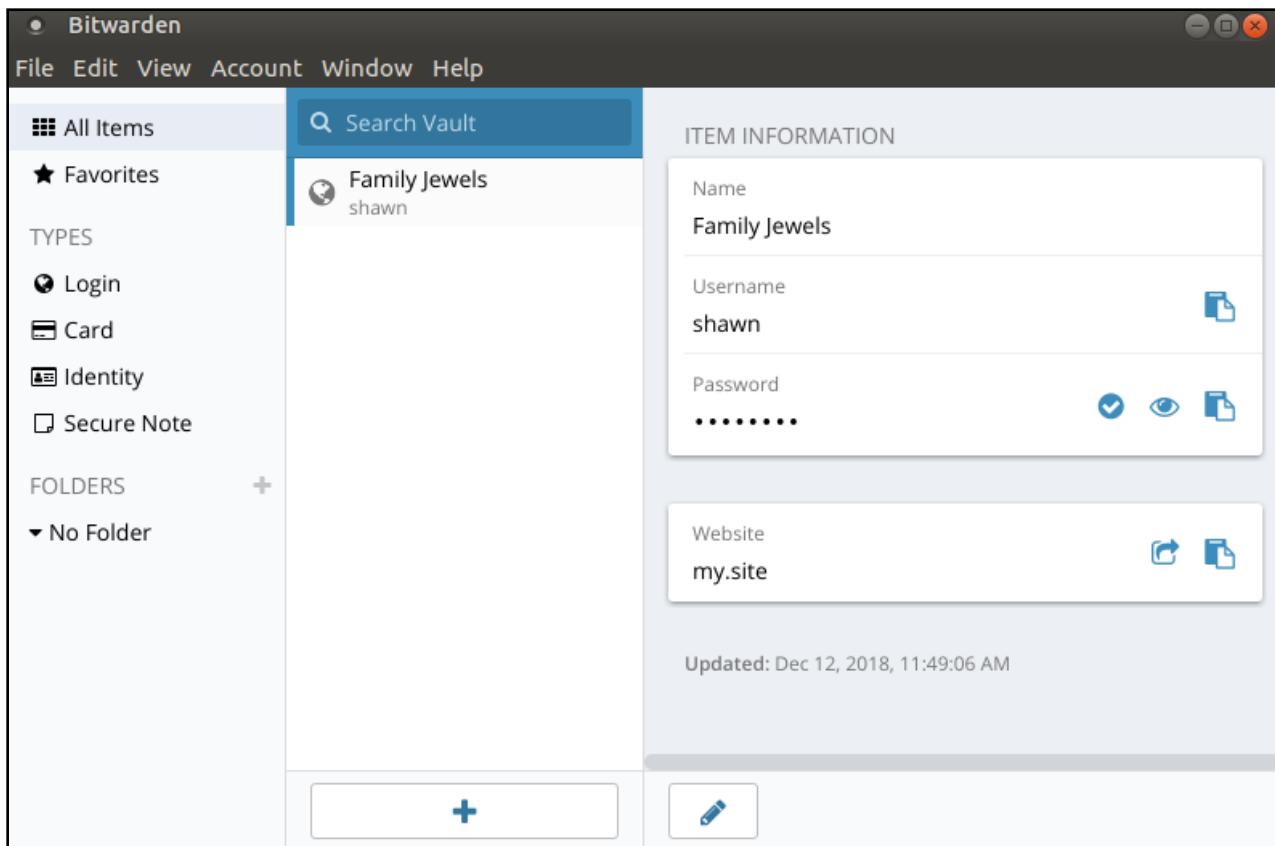


Figure 3. Bitwarden is very well designed, and with its open-source nature, it's hard to beat.

- Decent password generator.
- Native apps for Linux, Windows, Mac, Android and iOS.
- Browser extensions for all major browsers.
- Options to store logins, secure notes, credit cards and so on.

Bitwarden Pros:

- One developer for all apps.
- Open-source!

- Cloud-based access.
- Works offline if the “cloud” is unavailable.
- Free version isn’t crippled.
- Browser plugin works very well.

Bitwarden Cons:

- Database is stored in the cloud (again, it’s a pro and a con, depending).
- Some 2FA options require the Premium version.

Bitwarden Premium Version:

- \$10/year.
- Additional 2FA options.
- 1GB encrypted storage.

I’ll admit, Bitwarden is very, very impressive. If I had to pick a personal favorite, it probably would be this one. I’m already using a different option, and I’m happy with it, but if I were starting from scratch, I’d probably choose Bitwarden.

1Password

1Password is a widely used program for password management. But honestly, I’m not sure why. Don’t get me wrong; it works well, and it has great features. The problem is that I can’t find any features it has over the alternatives, and there isn’t a free option at all.

There’s also no native Linux application, but the 1PasswordX browser extension

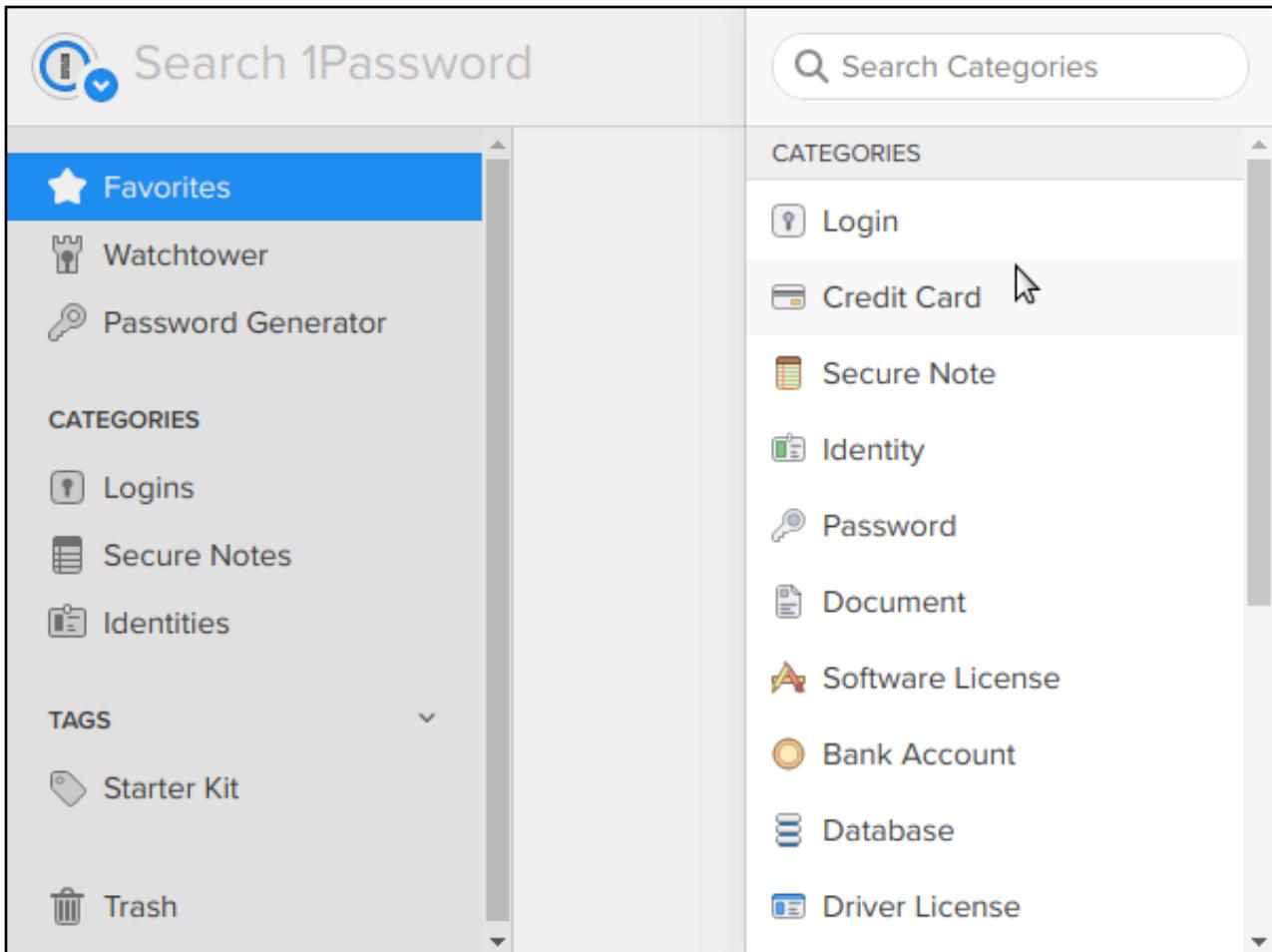


Figure 4. 1Password has a great interface, and it stores lots of data.

works well under Linux, and it's user-friendly enough to use for things other than browser login needs. Still, although I don't begrudge the company for charging a fee for the service, the alternatives offer significant services for free, and that's hard to beat. Finally, 1Password utilizes a "secret key" that's required on each device to log in. Although it is an additional layer of security, in practice, it's a bit of a pain to install on each device.

1Password Features:

- Cloud-based storage.

- Non-login data encryption (Figure 4).
- Printable “emergency kit” for recovering account.
- Cross-platform browser extension.
- Offline access.

1Password Pros:

- Easy-to-use interface.
- Very good browser integration.

1Password Cons:

- \$3/month, no free features.
- Secret-key system can be cumbersome.
- No native Linux app.
- Proprietary, closed-source code.

1Password Premium Features:

- All features require a monthly subscription.

If there weren’t any other password managers out there, 1Password would be incredible. Unfortunately for the 1Password company, there are other options, several of which are at least as good. I will admit, I really liked the browser extension’s interface, and it handled inserting login information into authentication fields very well. I’m not convinced it’s enough for the premium price, however, especially since

there isn't a free option at all.

LastPass

Okay, first I feel I should admit that LastPass is the password manager I use (Figure 5). As I mentioned previously, if I were to start over from scratch, I'd probably choose Bitwarden. That said, LastPass keeps getting better, and its integration with browsers, mobile devices and native operating systems is pretty great.

LastPass offers a free tier and a paid tier. Not too long ago, you had to pay for the premium service (\$2/month) in order to use it on a mobile device. Recently, however,

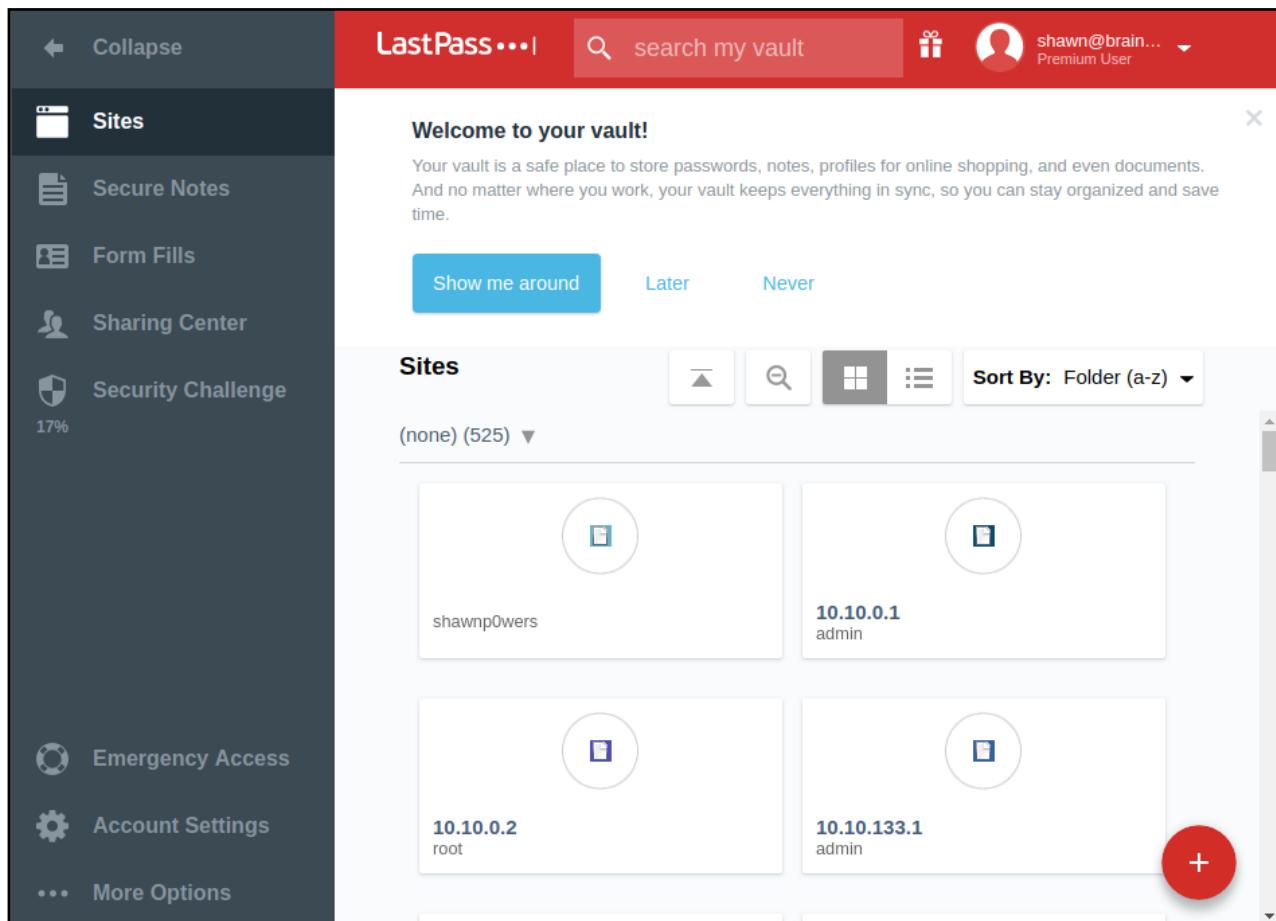


Figure 5. I seldom use anything other than LastPass's browser extension, unless I'm on my mobile device, but the app looks very similar.

LastPass opened mobile device syncing and integration into the completely free offering. That is significant, because it brings the free version to the same level as the free version of Bitwarden. (I suspect perhaps Bitwarden is the *reason* LastPass changed its free tier, but I have no way of knowing.)

LastPass Features:

- Cloud-based storage.
- Native apps for Linux, iOS and Android.
- 2FA.
- Offline access.
- Cross-platform browser extension.

LastPass Pros:

- Cloud-based storage.
- Very robust free offering.
- Smoothest browser-based password saving (in my experience).

LastPass Cons:

- Data stored in the cloud (yes, it's a pro and a con, depending).
- Rumored to have poor support (I've never needed it).
- Proprietary, closed-source code.

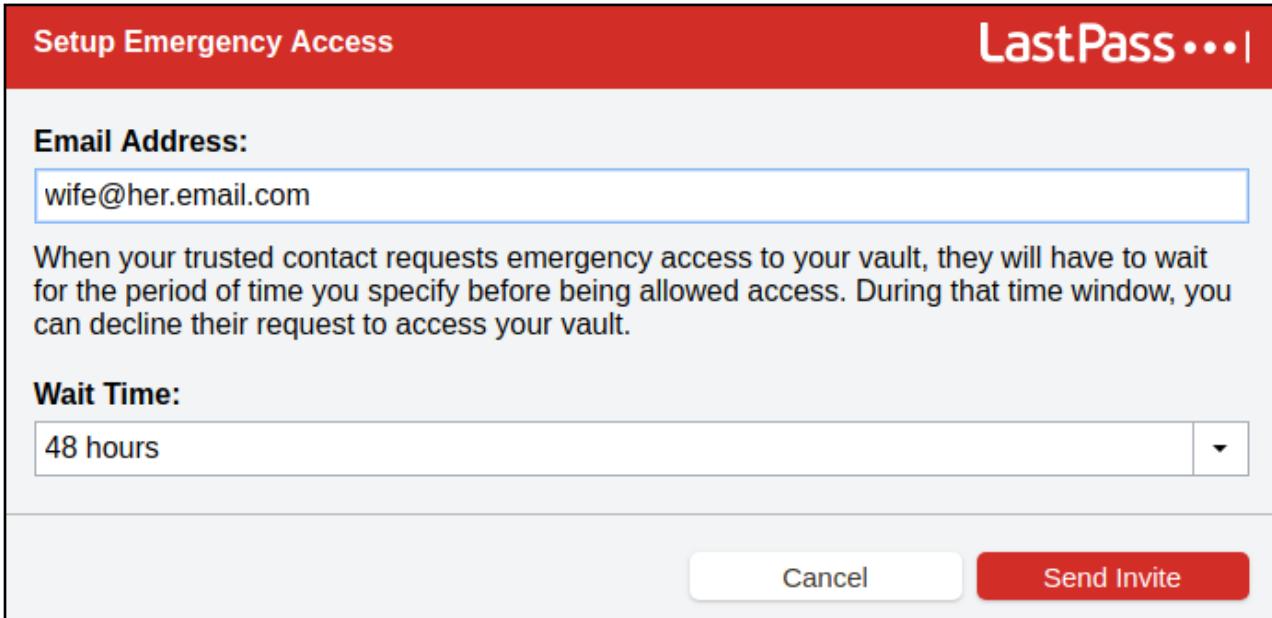


Figure 6. This is sort of a “deadman’s” switch for emergency access. It allows you to give emergency access to someone, with the ability to revoke that access before it actually happens. Pretty neat!

LastPass Premium:

- \$2/month.
- Gives 1GB online file storage.
- Provides the ability to share passwords.
- Enhanced 2FA possibilities.
- Emergency access granting (Figure 6).

LastPass is the only option I can give an opinion on based on extended experience. I did try each option listed here for a few days, and honestly, each one was perfectly acceptable. LastPass has been rock-solid for me, and even though it’s not

open source, it does work well across multiple platforms.

The Winner?

Honestly, with the options available, especially those highlighted today, it's hard to lose when picking a password manager. I sort of picked the top managers, and gave an overview of each. There are other, more obscure password managers. There are some options that are Linux-only. I decided to look at options that would work regardless of what platform you find yourself on now or even in the future. Once you pick a solution, migrating is a bit of a pain, so starting with something flexible is ideal.

If you're concerned about someone else controlling your data (even if it's encrypted), the KeePass/KeePassX/KeePassXC family is probably your best bet. If you don't mind trusting others with your data-syncing, LastPass or Bitwarden probably will be ideal. I suppose if you don't trust "free" products, or if you just really like the layout of 1Password, it's a viable option. And I guess, in a pinch, using browser password management is better than nothing. But please, be sure the data is encrypted and password-protected.

Finally, even if none of these options are something you'd use on a daily basis, consider recommending one to someone you care about. Keeping track of passwords in a secure, sync-able database is a huge step in living a more secure online lifestyle. Now that mobile devices are taken seriously in the password management world, password managers make sense for everyone—even your non-techie friends and family. ■



Shawn is Associate Editor here at *Linux Journal*, and has been around Linux since the beginning. He has a passion for open source, and he loves to teach. He also drinks too much coffee, which often shows in his writing. You can contact Shawn via e-mail, ljeditor@linuxjournal.com.

Resources

- [KeePass](#)
- [KeePassX](#)
- [KeePassXC](#)
- [1Password](#)
- [LastPass](#)
- [Bitwarden](#)

Send comments or feedback
via <https://www.linuxjournal.com/contact>
or email ljeditor@linuxjournal.com.

Everyday Security Tips

Make your computer safer with these guidelines based on the Linux Foundation's Security Checklist developed for corporate systems.

By Michael McCallister

It's an eternal problem. How can you best protect your computer from all the threats that being connected to a network brings? It's as much a concern for corporate system administrators as it is for everyday computer users. For Linux system administrators, it's one thing to protect a single system with permissions, but it's another matter altogether to protect a network when all your users can be physically located elsewhere.

Since 2015, the Linux Foundation has published a Security Checklist on GitHub, making the guidelines available to Linux admins everywhere. Sysadmins are encouraged to fork the checklist and then customize it for their own users. At the time of this writing in December 2018, 324 forks were recorded in the GitHub repository.

Although the guidelines are focused on laptops that connect to company servers, every Linux user who can sign in as root (that is, be a system administrator) will find useful tips and software recommendations to keep their systems safer, if not entirely bullet-proof.

In this article, I highlight some of the most important recommendations and provide some relevant tips from my own experience.

Hardware: It's Time to Reconcile with Secure Boot

Let's start with a brief history lesson. Computers nearly always have had an interface that allowed its firmware to hand over control of the system to humans. In the early days of IBM-compatible personal computing, that interface was called the Basic Input Output System (BIOS). This system worked just fine for a long time. There was a security problem with the BIOS, however. The BIOS didn't care whether the operating system that it was told was taking over was actually an operating system. Some folks developed malware, called rootkits, that seized control of the computer on which it was loaded, and the infected computer did the rootkit's bidding instead of the OS it was impersonating.

The solution (many years later) is the Unified Extensible Firmware Interface (UEFI), which began implementation in most Windows-based PCs in 2011. Among other things, UEFI contained the Secure Boot feature, requiring an operating system to present a digital signature that matched one in the UEFI database. Since Microsoft developed this standard and defined the approved signatures, Windows machines were preferred. For nearly two years after the standard was introduced, Linux installations often were rejected by Secure Boot. The Linux Foundation, among others, developed workarounds that allowed both replacement of pre-installed Windows and multibooting systems with a bootloader allowing users to select the OS at boot time.

Today, nearly every PC and laptop running UEFI and Secure Boot also can run Linux with little extra effort. The Linux Foundation checklist calls running with Secure Boot an essential piece of a secure Linux system. Chances are good that your laptop is running under the UEFI, as long as it was built after 2011.

If you're already running Linux on your laptop, look for the efi folder on your system:

```
ls /sys/firmware/efi/
```

If you're installing Linux on a new machine, you may want to check your

Anti Evil Maid

If you’re not persuaded, or for some reason you can’t get Secure Boot working with your distribution, the checklist offers an alternative called Anti Evil Maid (AEM). Created and maintained by the QubesOS project, Anti Evil Maid flips the authentication paradigm by “authenticating machine to the user”.

AEM runs on systems using dracut/initramfs, including (besides QubesOS) Fedora, Red Hat Enterprise, openSUSE, SUSE Linux Enterprise, Debian and Gentoo.

To get started with AEM, simply download the code from [GitHub](#) and review the setup instructions in the [README](#).

distribution’s documentation to confirm that it will boot under Secure Boot. [See *Kyle Rankin’s article “Tamper-Evident Boot with Heads”* for more on securing the boot process.]

Protecting UEFI

Secure Boot and UEFI are designed to protect the boot sector, so make sure that the UEFI configuration tool is password-protected. Manufacturers may limit the length of such a password, so check that as well.

If you’re lucky, you can set up the system to require the UEFI password to boot the system. This ability is considered “nice to have”, but not essential, especially if your disk is encrypted, and you shut down/restart the machine regularly.

Encrypt, Encrypt, Encrypt

Say you’re ready to install Linux on a machine. There are two things to check before choosing a distribution:

1. It should support full-disk encryption.
2. It should support either Mandatory Access Controls (MAC) or Role-Based Access Controls (RBAC), usually managed through SELinux or AppArmor.

All of your partitions (home, root and swap) will carry a big pile of sensitive data that likely will make some evil-doer very happy should it get loose. If your system can use the bootloader from an encrypted /boot partition, you should do that too.

The reigning standard for Linux disk encryption is the Linux Unified Key System (LUKS). This system, including the dm-crypt package, encrypts the disk with the strong AES-256 encryption algorithm. It also supports multi-user logins into the operating system.

Ideally, your selected distribution will include dm-crypt/LUKS as part of its installation package. If not, ensure that it's available in a repository. The package has different names, but searching for "LUKS" in your package manager should find the right package. If you encrypt your disk after initial system install, be sure to back up your system to a remote location first, as dm-crypt will wipe the existing partitions!

SELinux and AppArmor

Security Enhanced Linux (SELinux) was originally developed by the US National Security Agency (NSA) in 2000 and merged into the Linux kernel in 2003. SELinux is a Mandatory Access Controls (MAC) system designed to extend core permissions. For those suspicious of the NSA's intent, the code is released under the GNU General Public License. The system was considered to be more of a hindrance by many at the beginning, but the Linux Foundation guidelines declare that it's "mature, robust, and has come a long way since its initial rollout".

Nonetheless, the Linux Foundation says that SELinux "will have limited security benefits on the workstation", because most desktop-type applications an

Reminder about Strong Passphrases

The checklist recommends creating and using two “distinct, robust, equally strong passphrases” if you are the only user of your machine. One passphrase is for your administrative tasks (unlocking your encrypted disk, bootloader and other root tasks), and a separate passphrase is for your user account and also is to open your password manager for all the other accounts you have.

If someone else uses your machine, you should have a third passphrase for unlocking the disk. That way, you can continue to manage the machine as root, but your ordinary users can unlock their own files.

You still may pause when reading about “passphrases”. You know about passwords, and you know (at least in theory) that p@ssw0rd really doesn’t keep anything safe. The problem is that the more secure your passwords are, the harder they are to remember. Passphrases—a series of words that you can memorize that no one else can crack—are so much better!

Don’t forget Passphrase Rule #1: don’t use “The future is already here—it’s just not evenly distributed” or some other phrase that you think is cool, geeky but obscure. Chances are the phrase isn’t so obscure. If you want to use a quote from classic literature, you might choose “Future is obscure more like a granfalloon” instead.

ordinary user runs would be unconfined. The Linux Foundation recommendation is *don’t turn off SELinux if you run Red Hat or another distribution that uses SELinux for security, but leave its default settings alone.*

Many distributions prefer AppArmor to enhance Linux system security. This Role-Based Access Control (RBAC) is easier to configure than SELinux, and it will better protect your network-facing applications.

Post-Install Hardening

Once you have installed your Linux distribution, and your data and Swap partitions are encrypted, you should tweak a few more settings to get the system in better shape.

Obsolete kernel modules: FireWire and Thunderbolt standards were created to make it easier for multiple devices to connect to a laptop or desktop system. They, along with ExpressCard, allowed connecting devices to have full direct memory access to your system—really nice when you’re the only one trying to access the machine with another device. It’s not so good when attackers are everywhere, including unprotected coffee-shop WiFi.

Since more secure technologies have been developed, you don’t need these kernel modules. Although newer machines may not have these ports (and you’re better off if they don’t), you should blacklist those kernel modules. Open /etc/modprobe.d/blacklist-dma.conf and add these lines:

```
blacklist firewire-core  
blacklist thunderbolt
```

Readable root mail: your system sets up mailboxes for every user on the system, and that email box is likely open at username@systemname. By “every user”, I include root. The system automatically sends messages to root’s system email. These messages can include security reports and other important notifications. You may be like me and not add that mailbox to your email reader, so the mailbox fills up with important unread email.

Are you suddenly getting the feeling that you may not know what horrible security breaches may have already happened that you don’t even know about? You can fix this relatively easily. Forward system mail to an account that is in your email reader. Open /etc/aliases (as root) in your favorite text editor, and add these recommended lines:

```
#Person who should get root's email  
root: mike@example.com
```

Close incoming sshd ports: Secure SHell (SSH) is a wonderful thing, but you probably don't need to allow other machines (even with a passphrase!) to connect to this laptop. Without a really good reason to allow incoming secure connections, don't do it. Check your firewall settings, as they vary from distribution to distribution, and filter out incoming sshd requests. Then disable the sshd service with these systemd commands:

```
systemctl disable sshd.service  
systemctl stop sshd.service
```

Updates: I'm obsessive about updating my software, partly because I've always wanted the latest and greatest stuff, but also because updates fix bugs and security holes. Although I have my complaints with KDE's Discover update service, I still am almost gleeful when I'm told I have updates ready to install.

You may be one of those people who had an automatic update go badly, rendering your system useless. Perhaps you update software on a case-by-case basis, reading carefully what exactly an update is intended to fix. You read carefully every security bulletin that comes out on your system, and update accordingly.

The Linux Foundation checklist recommends turning on automatic updates, but if you fall into the camp of wary updaters, you should at least permit automatic *notifications* of available updates. Combine that with immediate response to any security bulletins you receive from your distribution, and you should be in good shape.

Backups

As a security-conscious Linux user, you probably have a backup regimen already in place. In case you don't, the checklist offers a set of guidelines you should be able to follow:

- Critical directories to back up: /home, /etc, and /var/log.

- Ideally, you have an external (LUKS-encrypted) hard drive to copy over those directories to on a regular basis.
- Don’t ever put your home directory on an unencrypted medium, even as a temporary measure!
- If you haven’t encrypted your backup disk, your backup tool should be able to encrypt your backups. The shell program Duplicity and its deja-dup GUI counterpart should work nicely.
- Cloud or otherwise offsite backups should focus on the most important files to avoid shoving huge amounts of data over the internet.
- Consider using SpiderOak for managing offsite content.

Suggestions for Desktop Users

Okay, Linux is installed, and it’s as secure as you can make it. You have a backup strategy in place, so you can protect your work if something bad happens. Now you’re ready for day-to-day use of your Linux computer. You’re still not done with security. As the checklist puts it, “The world of IT security is a rabbit hole with no bottom.”

As a “non-exhaustive list”, the Linux Foundation offers a set of best practices for day-to-day use:

- Dump the X Window System for Wayland. As the World Wide Web protocols were not conceived with security in mind, so X11 (which is much older than the web) is much more about connecting people and machines than preventing bad actors from wreaking havoc. It’s taking a while for distributions to transition to Wayland, but consult with your distribution to see how you can default to Wayland yourself.
- Use a password manager. Most browsers prompt you whether you want to

save a site password after you enter credentials the first time. These are okay, but that won't help if you use a different browser the next time you visit that site. The Linux Foundation recommends a standalone, cross-browser password manager, such as KeePassX, Pass, Django-Pstore or (if you're using Puppet for infrastructure) Hiera-Eyaml.

- Use Fido U2F USB tokens to provide two-factor authentication. The FIDO group works to eliminate passwords for authentication altogether. This hardware-based two-factor authentication system is a first step. Go [here](#) for a list of services supporting U2F.
- Secure your private keys! Public Key Infrastructure (PKI) breaks down if anyone other than you gains access to your SSH and PGP private keys. Protect them with strong passphrases.
- Don't suspend your machine at the end of the day. It's much better to shut it down or put it in hibernation if you're going to be away from the box for a long time.

Using Multiple Browsers

The Linux Foundation checklist expresses concern that web browsers offer "the largest and most exposed attack surface on your system. It is a tool written specifically to download and execute untrusted, frequently hostile code." As an essential means to mitigate the size of this attack surface, the checklist recommends using two browsers for all online activity:

1. Use Firefox for work and high-security sites. The Linux Foundation suggests that Firefox should be used when you want to ensure that data like cookies, sessions, login information and keystrokes aren't captured by attackers.
2. Use a Chromium-based browser for everything else. Give it a distinctive theme from Firefox for a visual cue that this is your browser for untrusted sites. Development on this project is further along than Firefox for security features

like seccomp sandboxes and kernel user namespaces. These features “act as an added layer of isolation between the sites you visit and the rest of your system”.

Install Privacy Badger, HTTPS Everywhere and uMatrix plugins on both browsers. Add Firejail on Firefox for sandbox protection. These plugins make it less likely that your browsing data is being collected across the web. uMatrix is apparently so effective at preventing active third-party content from loading on your screen, it makes it a bit of a hassle to use the web at all, so install it with caution!

I use Privacy Badger and HTTPS Everywhere on Firefox, and I feel like that’s enough to protect me on the web. I’ll continue to think about this last recommendation (using browsers based on what I’m doing). Interesting to consider though.

Conclusion

In this article, I’ve gone over the “essential” recommendations on the checklist and some of the “nice” options. The checklist offers a few items for the *paranoid* among us as well. Be sure to check out the [whole list](#), with more links and discussion points, and see also the Resources section for this article. ■

Michael McCallister has written about Linux and FLOSS since the turn of the millennium. Find him at michaelmccallister.com, Author, MichaelMcCallister on Facebook, and @workingwriter at Twitter and most everywhere else online.

Resources

- [Linux workstation security checklist](#)
- [UEFI](#)
- [UEFI Secure Boot](#)
- [SELinux](#)
- [AppArmor](#)
- [Duplicity](#)
- [SpiderOak](#)
- [USB-Dongle Authentication](#)
- [KeePassX](#)
- [pass \the standard unix password manager](#)
- [django-pstore](#)
- [Hiera-Eyaml](#)
- [Privacy Badger](#)
- [HTTPS Everywhere](#)
- [uMatrix](#)
- [Firejail Security Sandbox](#)

Send comments or feedback
via <http://www.linuxjournal.com/contact>
or email ljeditor@linuxjournal.com.

Understanding Public Key Infrastructure and X.509 Certificates

An introduction to PKI, TLS and X.509, from the ground up.

By Jeff Woods

Public Key Infrastructure (PKI) provides a framework of encryption and data communications standards used to secure communications over public networks. At the heart of PKI is a trust built among clients, servers and certificate authorities (CAs). This trust is established and propagated through the generation, exchange and verification of certificates.

This article focuses on understanding the certificates used to establish trust between clients and servers. These certificates are the most visible part of the PKI (especially when things break!), so understanding them will help to make sense of—and correct—many common errors.

As a brief introduction, imagine you want to connect to your bank to schedule a bill payment, but you want to ensure that your communication is secure. “Secure” in this context means not only that the content remains confidential, but also that the server with which you’re communicating actually belongs to your bank.

Without protecting your information in transit, someone located between you and your bank could observe the credentials you use to log in to the server, your account information, or perhaps the parties to which your payments are being sent. Without being able to confirm the identity of the server, you might be surprised to learn that

you are talking to an impostor (who now has access to your account information).

Transport layer security (TLS) is a suite of protocols used to negotiate a secured connection using PKI. TLS builds on the SSL standards of the late 1990s, and using it to secure client to server connections on the internet has become ubiquitous. Unfortunately, it remains one of the least understood technologies, with errors (often resulting from an incorrectly configured website) becoming a regular part of daily life. Because those errors are inconvenient, users regularly click through them without a second thought.

Understanding the X.509 certificate, which is fully defined in RFC 5280, is key to making sense of those errors. Unfortunately, these certificates have a well deserved reputation of being opaque and difficult to manage. With the multitude of formats used to encode them, this reputation is rightly deserved.

An X.509 certificate is a structured, binary record. This record consists of several key and value pairs. Keys represent field names, where values may be simple types (numbers, strings) to more complex structures (lists). The encoding from the key/value pairs to the structured binary record is done using a standard known as ASN.1 (Abstract Syntax Notation, One), which is a platform-agnostic encoding format.

Designed for efficient operation across a broad range of hosts, ASN.1 allows several ways to encode values using its “basic encoding rules” (BER). As you will see shortly, multiple encodings for the same data will not work for your certificates, so the X.509 standard designated a subset of BER as the “distinguished encoding rules” (DER). The use of DER ensures that there is exactly one way any value might be encoded.

A certificate may be distributed in this raw, binary DER format. Since binary data is not terminal- or email-friendly, the encodings defined in the Privacy Enhanced Mail (PEM) standard are commonly applied. Although PEM is largely an obsolete standard, it defines methods of encoding binary material in a text-safe format. Most important, it defines the base64 scheme for encoding binary data into text and specifies the use of encapsulation boundaries to signal the beginning and

Encryption: Symmetric vs. Asymmetric

Two broad types of encryption are widely used to secure web traffic: symmetric key and asymmetric key.

In both types of encryption, the key refers to a passphrase (of sorts) required to encrypt or decrypt the data. Without the key, an entity attempting to read the data would be unable to read the encrypted content. Likewise, malicious content would be difficult to generate.

Symmetric key encryption uses the same key on both sides of the communication channel to encrypt or decrypt data. Symmetric key encryption is implemented in algorithms such as AES or DES. It has the advantage of being very fast with a low overhead, but a secure channel must exist between the two parties through which the key may be exchanged.

Asymmetric key encryption uses a pair of keys, known as a private key and a public key. These keys are different values. Data encrypted using the private key can be decrypted only using the public key. The reverse is also true: data encrypted with the public key can be decrypted only with the private key. Asymmetric encryption algorithms include algorithms such as RSA, DSA and ECDSA.

While symmetric keys have desirable properties for communication, they must be generated and exchanged in a secure manner. Asymmetric algorithms fit this niche, and you can use them as the foundation for building a secured channel.

ending of encoded content. The use of the PEM format is generally preferred on Linux servers (more on that later in this article).

Certificates may be distributed in a multitude of other formats as well. What is important to understand is that regardless of the encoding format—DER, PEM, PFX or something else—all certificates are basically the same when they are decoded. Tools,

such as OpenSSL, are able to read or convert any of these formats easily.

The certificate encodes two very important pieces of information: the server’s public key and a digital signature that can be used to confirm the certificate’s authenticity. Additionally, the certificate will include metadata used by the CA to track the certificate and provide guidelines on how the public key can be used.

Public key cryptography, also known as asymmetric key cryptography, provides a mechanism to establish a secured communication channel over an insecure network. Using the server’s public key, the client and server are able to negotiate a shared symmetric key securely, which can be used to secure communications.

But, how do you (as the client) know that the public key can be trusted as authentic? You can use the certificate authority (CA), a trusted third party, as a mediator of sorts. By submitting the certificate to be signed by the CA, the owner of the server gives consent for the certificate to be “authenticated” by the CA. If the client, who trusts that the CA has verified the server properly, can confirm that the signature is valid, the certificate can be trusted.

A hash, often using the SHA256 algorithm, is a digital fingerprint of the data. If you change a single bit in the data, the hash will change. By computing a hash over the DER-encoded public key section of the certificate and then signing the hash with its own private key, the CA is giving its stamp of approval on the certificate. This signed hash value is the signature appended to the certificate.

When the client receives the server’s certificate, the client can create the hash over the same data in the certificate that was signed by the CA. If the client is able to decrypt the signature using the CA’s public key and it matches the hash the client computed itself, the client can be certain that the server’s certificate was endorsed by the CA.

The final piece of the puzzle is understanding that the client must trust the CA explicitly. This is done by adding the CA’s public key to the client’s “trusted key store”.

As the user of common web services, you don't need to think much about certificate authorities. Both your operating system and your web browser ship with curated lists of authorities, which have been pre-selected as trustworthy for you. Still, from time to time, it may be necessary to install a certificate from a non-standard CA.

Now that you understand the basics of how a certificate is put together, let's look at a real-world example. Using the following command, you can pull the certificate from the google.com domain:

```
$ openssl s_client -showcerts -connect google.com:443 </dev/null
```

This command connects to the remote server using TLS and dumps a large amount of information about the TLS handshake to your console. If you had not redirected standard input from /dev/null, the connection would remain open, allowing you to interact directly with the server on port 443 (you can close this connection explicitly with Ctrl-D).

Below is an (abbreviated) view of the output, which I explore through the remainder of this article:

```
$ openssl s_client -showcerts -connect google.com:443
</dev/null
CONNECTED(00000003)
depth=2 OU = GlobalSign Root CA - R2, O = GlobalSign, CN =
    GlobalSign
verify return:1
depth=1 C = US, O = Google Trust Services, CN = Google
    Internet Authority G3
verify return:1
depth=0 C = US, ST = California, L = Mountain View, O =
    Google LLC, CN = *.google.com
verify return:1
---
```

Certificate chain

```
0 s:/C=US/ST=California/L=Mountain View/O=Google
↳ LLC/CN=*.google.com
    i:/C=US/O=Google Trust Services/CN=Google Internet
    ↳ Authority G3
-----BEGIN CERTIFICATE-----
MII IgjCCB2qgAwIBAgIITFQTbb/xK/QwDQYJKoZIhvcNAQELBQAwVDELMAkGA1UE
BhMCVVVMxHjAcBgNVBAoTFUdvb2dsZSBUcnVzdCBTZXJ2aWNlczElMCMGA1UEAxMc
R29vZ2xlIEludGVybmV0IEF1dGhvcml0eSBHMzAeFw0xODEwMzAxMzE1MDVaFw0x
    <... content omitted ...>
OTAxMjIxMzE1MDBaMGYxCzAJBgNVBAYTA1VTMRMwEQYDVQQIDA�DYWxpZm9ybmlh
pvxy sdjrJ8qfUyD0AY/Z8dCs1RQfx8SKbXuoML9e0X5uxRmeyjQ0s+BPJDIQG5b8
IGSRGSm8vWtg9vz/GDZIErtE01kgX0s1BBGL5NSCFpxkp1lh/Usi3nFzPcU6Fbx
WMSdoZZKpgy5+6GGjYv/dEyEdnXYzg==
-----END CERTIFICATE-----
1 s:/C=US/O=Google Trust Services/CN=Google Internet
    ↳ Authority G3
    i:/OU=GlobalSign Root CA - R2/O=GlobalSign/CN=GlobalSign
-----BEGIN CERTIFICATE-----
MII EXDCCA0SgAwIBAgINAe0pMBz8cgY4P5pTHTANBgkqhkiG9w0BAQsFADBMMSSAw
HgYDVQQLExdHbG9iYWxTaWduIFJvb3QgQ0EgLSBSMjETMBEGA1UEChMKR2xvYmFs
U2lnbjETMBEGA1UEAxMKR2xvYmFsU2lnbjAeFw0xNzA2MTUwMDAwNDJaFw0yMTEy
    <... content omitted ...>
FIwsI0NGl1p3A8CgxkqI/UAih3JaG0qcpcdaCIzkBaR9uYQ1X4k2Vg5APRLouzVy
7a8IVk6wuy6pm+T7HT4LY8ibS5FEZlfAFLSW8NwsVz9SBK2Vqn1N0PIMn5xA6NZV
c7o835DLAFshEWfC7TIe3g==
-----END CERTIFICATE-----
---  

    <... content omitted ...>  

---  

DONE
```

This output shows that the server returned not one, but two PEM-encoded

certificates. Each certificate is bracketed by -----**BEGIN CERTIFICATE**----- and -----**END CERTIFICATE**----- markers. Although they look intimidating, remember that they are nothing more than base64-encoded DER data, and you have tools that can look inside!

OpenSSL provides a little bit of context just above the **BEGIN** mark of each certificate with the **s:** (subject) and **i:** (issuer) tags. The subject tells you what server (or other entity) the certificate was generated for, while the issuer tells you which certificate authority signed the certificate. If the subject matches the server to which you are connecting and you trust the issuer, you can be on your way.

At this point, let's pause to note that the X.509 standard used to encode certificates descends from the same series of X.500 standards as LDAP. You can see some of this common lineage in the directory syntax used to identify the subject and issuer:

C=US,ST=California,L=Mountain View,O=Google LLC,CN=*.google.com

If you're not familiar with LDAP naming standards, it's most important to understand that the "CN" is the "common name" of the certificate owner, with the remainder of the name being used for organization within a directory.

You can decode these certificates using OpenSSL. Extract the first certificate (**CN=*.google.com**) into a file named "google_com.crt" and the second certificate (**CN=Google Internet Authority G3**) into a file named "google_authority_g3.crt". Include the **BEGIN** and **END** wrapping each certificate, but nothing more. When you're done, you should have two files that look like this:

```
$ cat google_com.crt
-----BEGIN CERTIFICATE-----
MII IgjCCB2qgAwIBAgIITFQTbb/xK/QwDQYJKoZIhvcNAQELBQAwVDELMAkGA1UE
BhMCVVVMxHjAcBgNVBAoTFUdvb2dsZSBUcnVzdCBTZXJ2aWNlczElMCMGA1UEAxMc
R29vZ2xlIEludGVybmb0IEF1dGhvcmloeSBHMzAeFw0xODEwMzAxMzE1MDVaFw0x
<... content omitted ...>
```

```
OTAxMjIxMzE1MDBaMGYxCzAJBgNVBAYTA1VTMRMwEQYDVQQIDA�DYWxpZm9ybmlh
pvxysdjrJ8qfUyD0AY/Z8dCs1RQfx8SKbXuoML9e0X5uxRmeyjQ0s+BPJDIQG5b8
IGSRGSm8vWtg9vz/GDZIErtE01kgXOs1BBGL5NSCFpxkp1lh/Usi3nFzPcU6Fbxv
WMSdoZZKpgy5+6GGjYv/dEyEdnXYzg==
```

-----END CERTIFICATE-----

```
$ cat google_authority_g3.crt
```

-----BEGIN CERTIFICATE-----

```
MIIEXDCCA0SgAwIBAgINAe0pMBz8cgY4P5pTHTANBgkqhkiG9w0BAQsFADBMMSSAw
HgYDVQQLExdHbG9iYWxTaWduIFJvb3QgQ0EgLSBSMjETMBEGA1UEChMKR2xvYmFs
U2lnbjETMBEGA1UEAxMKR2xvYmFsU2lnbjAeFw0xNzA2MTUwMDAwNDJaFw0yMTEy
<... content omitted ...>
```

```
FIwsI0NGl1p3A8CgxrqI/UAih3JaG0qcpcdaCIzkBaR9uYQ1X4k2Vg5APRLouzVy
7a8IVk6wuy6pm+T7HT4LY8ibS5FEZlfAFLSW8NwsVz9SBK2Vqn1N0PIMn5xA6NZV
c7o835DLAFshEWfC7TIe3g==
```

-----END CERTIFICATE-----

Now you can decode the content of either certificate using the command:

```
$ openssl x509 -in google_com.crt -noout -text
```

Here's a slightly edited version of the output:

Certificate:

Data:

Version: 3 (0x2)

Serial Number: 5500042407018834932 (0x4c54136dbff12bf4)

Signature Algorithm: sha256WithRSAEncryption

Issuer: C = US, O = Google Trust Services, CN = Google

 ↳Internet Authority G3

Validity

Not Before: Oct 30 13:15:05 2018 GMT

Not After : Jan 22 13:15:00 2019 GMT

Subject: C = US, ST = California, L = Mountain View,
↳0 = Google LLC, CN = *.google.com

Subject Public Key Info:

 Public Key Algorithm: rsaEncryption

 Public-Key: (2048 bit)

 Modulus:

 00:d1:bf:94:10:1f:94:15:bd:6c:3b:83:97:49:29:
 ad:08:63:18:11:1b:57:7d:4d:b3:3f:9c:cd:62:ed:
 eb:4d:d2:6b:78:3f:3f:01:48:43:a8:81:b6:42:f6:
 <... content omitted ...>
 e1:e4:24:b8:21:c4:9e:e5:86:c6:73:45:4f:a8:6f:
 e0:81:f3:4e:46:03:3d:e9:d2:01:5b:6f:57:3c:22:
 d4:83

 Exponent: 65537 (0x10001)

X509v3 extensions:

 X509v3 Extended Key Usage:

 TLS Web Server Authentication

 X509v3 Subject Alternative Name:

 DNS:*.google.com, DNS:*.android.com,
↳<... content omitted ...>

 X509v3 CRL Distribution Points:

 Full Name:

 URI:<http://crl.pki.goog/GTSGIAG3.crl>

Signature Algorithm: sha256WithRSAEncryption

 c7:57:a4:97:ad:32:e1:5f:10:53:05:ba:03:c4:cd:2e:11:c9:
 7d:36:a9:4c:16:a8:46:a1:5a:30:c4:4f:04:86:8d:8b:e1:95:
 24:34:62:94:48:b9:8a:3d:d2:d7:49:eb:a5:6c:59:72:c3:64:
 <... content omitted ...>
 16:9c:64:a7:59:61:fd:4b:22:de:71:73:3d:c5:3a:15:bb:f1:
 58:c4:9d:a1:96:4a:a6:0c:b9:fb:a1:86:8d:8b:ff:74:4c:84:
 76:75:d8:ce

This looks like progress—you can see the certificate provided by the google.com server in all of its glory! Let’s visit the highlights.

Near the top of the certificate, you can see the serial number in the “Data” section. The certificate authority gives each certificate a unique serial number when it is generated. This allows certificates to be identified uniquely if there ever is a need to revoke them.

Two “Signature Algorithm” blocks follow the Data block. The first of these contains the server’s public key information along with any extensions (options) enabled by the CA. The second block contains the signed hash generated by the CA.

Looking at the detail within the first “Signature Algorithm” block, you can see the subject, issuer and validity dates for the certificate. In general, the certificate may be used only on a server with a name matching the subject and within the dates specified. The “Issuer” (a CA) identifies the public key used to validate the signature in the second block. If this issuer’s public key is in the trusted key store, the certificate is valid.

Perhaps most important, you can see the server’s public key in this section. This is the key the client will use to encrypt content so that it can be decrypted only on the server. In this case, the server is providing a 2048-bit RSA key.

Below the server’s public key, you’ll find a block labeled “X509v3 extensions”. The extensions in this section are set by the CA when the certificate is signed and can be used to enable (or restrict) the use of the certificate. The full details of these extensions are defined in RFC 5280, but I’ll cover the highlights here.

You can see in the “X509v3 Extended Key Usage” section that the certificate is authorized for “TLS Web Server Authentication”. This means that the certificate may be used to identify a web server positively. Other common uses that might be listed here include functioning as a CA (allowing the signing of

certificates for other servers) or authorizing the certificate to be used as proof of a client’s identity.

The “Subject Alternative Names” section is optional, but it may be used to list any additional web servers permitted to use this certificate. The SAN section allows servers to be listed using either DNS names or IP addresses. As you can see in the example, wild cards are supported. If the certificate’s subject does not match the server name, but a match can be made here, the certificate is treated as if the subject were a match.

The final entry to look at in the X509v3 extensions block is the “X509v3 CRL Distribution Points”. A CRL is a “certificate revocation list”. These lists contain the serial numbers of any certificates issued by the CA that have been compromised, retired or (for some other reason) made invalid.

As a regular step in negotiating a TLS connection, the client should verify that the certificate presented by the server—or the serial numbers of a certificate used as a CA—does not appear on a CRL. Ideally, these serial numbers would be checked against a current version of the domain’s CRL each time a certificate is verified. In practice, pulling an updated CRL for each connection adds a lot of overhead to the process. Most clients will use a cache, which is rarely up to date but avoids issues if the site hosting the CRL is unavailable.

The second “Signature Algorithm” block contains the signed hash of the DER-encoded X.509 data found in the first block. You can see that the SHA256 hash algorithm was used, and that an RSA private key was used to sign the hash. After being decoded with the CA’s corresponding public key, this hash must match the value computed by the client for the same data.

This is where the use of DER for encoding is important: if there were multiple ways to encode the data in the certificate, as there might be using BER, the hash might assume several different values. By using DER, you guarantee that the values are encoded and decoded consistently into the same series of bytes. If a

single byte changes, a different hash would be created and the verification of the signature could not be completed!

Certificates contain a lot of detail, and all of it must line up before a client can trust the certificate. It shouldn't be surprising that things fall apart as often as they do.

The most common errors with TLS connections include:

- Server name mismatch, meaning that the certificate's CN (or any entries in the subject alternative names section) doesn't match the host that presents it.
- A certificate has expired and needs to be re-issued by the CA.
- The root CA used to sign the certificate is not in the client's trusted key store.

With the knowledge you've gained about certificates in this article, the meanings and (perhaps more important) solutions to those problems should become more clear. ■

Jeff Woods has worked in the IT field for more than 20 years, with broad experience in areas including software engineering, data engineering, operations, security engineering and DevOps. His experience with Linux dates back to 1993, when he began working with the SLS distribution. He currently works as an IT Architect for a global FinTech near Atlanta, Georgia. In addition to his IT experience, Jeff is a Navy veteran, Dad, and has served as a leader in various BSA programs for the past 12 years. You can contact him at jcwoods@gmail.com or via LinkedIn at <https://www.linkedin.com/in/jeff-woods-a50b921>.

Resources

- [Introduction to ASN.1](#)
- [RFC 5280 \(X.509 standard\)](#)
- [RFC 1421 \(Privacy Enhanced Mail\)](#)
- OpenSSL man pages relating to x509 manipulation, specifically `man x509` or `man openssl-x509`.
- OpenSSL man pages relating to secure client, specifically `man s_client` or `man openssl-s_client`.

Send comments or feedback
via <https://www.linuxjournal.com/contact>
or email ljeditor@linuxjournal.com.

WebAuthn Web Authentication with YubiKey 5

A look at the recently released YubiKey 5 hardware authenticator series and how web authentication with the new WebAuthn API leverages devices like the YubiKey for painless website registration and strong user authentication.

By Todd A. Jacobs

I covered the YubiKey 4 in the May 2016 issue of *Linux Journal*, and the magazine has published a number of other articles on both YubiKeys and other forms of multi-factor authentication since then. Yubico recently has introduced the YubiKey 5 line of products. In addition to the YubiKey's long-time support of multiple security protocols, the most interesting feature is the product's new support for FIDO2 and WebAuthn.

WebAuthn is an application programming interface (API) for web authentication. It uses cryptographic “authenticators”, such as a YubiKey 5 hardware token to authenticate users, in addition to (or even instead of) a typical user name/password combination. WebAuthn is currently a World Wide Web Consortium (W3C) candidate recommendation, and it's already implemented by major browsers like Chrome and Firefox.

This article provides an overview of the YubiKey 5 series, and then goes into detail about how the WebAuthn API works. I also look at how hardware tokens, such as

the YubiKey 5 series, hide the complexity of WebAuthn from users. My goal is to demonstrate how easy it is to use a YubiKey to register and authenticate with a website without having to worry about the underlying WebAuthn API.

About the YubiKey 5 Series

The YubiKey 5 series supports a broad range of two-factor and multi-factor authentication protocols, including:

- Challenge-response (HMAC-SHA1 and Yubico OTP).
- Client to Authenticator Protocol (CTAP).
- FIDO Universal 2nd-Factor authentication (U2F).
- FIDO2.
- Open Authorization, HMAC-Based One-Time Password (OATH-HOTP).
- Open Authorization, Time-Based One-Time Password (OATH-TOTP).
- OpenPGP.
- Personal Identity Verification (PIV).
- Web Authentication (WebAuthn).
- Yubico One-Time Password (OTP).

In addition, the entire YubiKey 5 series (with the exception of the U2F/FIDO2-only Security Key model) now supports OpenPGP public key cryptography with RSA key sizes up to 4096 bits. This is a notable bump from the key sizes supported by some earlier models. Yubico's OpenPGP support also includes an additional slot for an OpenPGP authentication key for use within an SSH-compatible agent, such as GnuPG's **gpg-agent**.

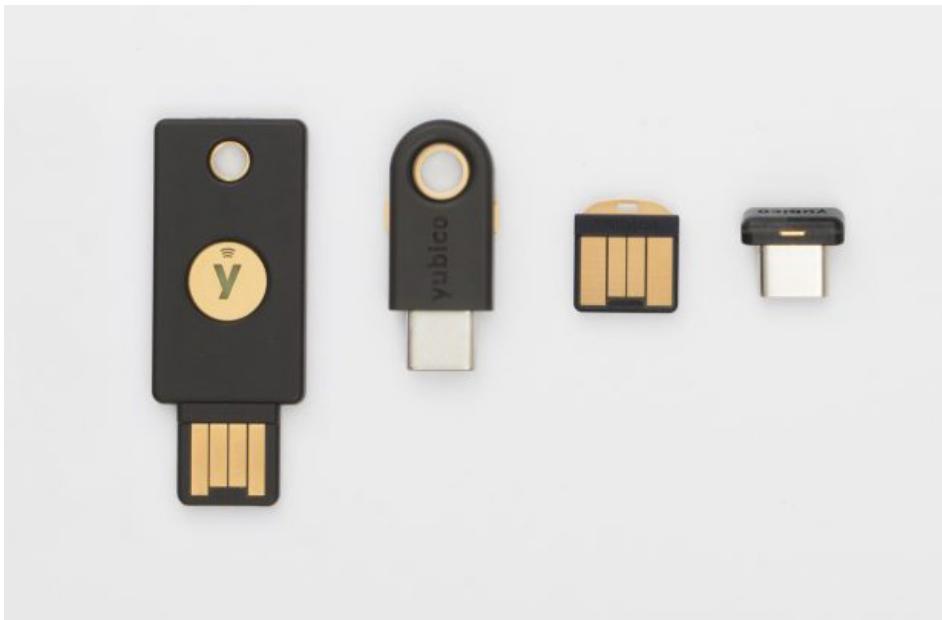


Figure 1.
YubiKey 5 Series



Figure 2. YubiKey 5 with NFC

You can create OpenPGP keys on your computer and then move them into the YubiKey, or you can create them directly on the key itself for extra security. On-key generation of crypto keys ensures that secret keys never are exposed to a host computer. This provides the ability to sign and decrypt data in relative safety, even when attached to systems that shouldn't be trusted with secret key material.

The YubiKey 5 series is also notable for its broad array of form factors, including USB-A, USB-C and low-profile “nano” devices that can be left in a port indefinitely. A number of form factors also support Near-Field Communication (NFC) for use with tablets and mobile phones.

The YubiKey 5 series also includes a low-cost model (aptly named the “Security Key”) in a USB-A form factor. The Security Key provides only U2F and FIDO2 authentication. This model is currently less than half the cost of its more capable brethren in the



Figure 3.
Security Key
by Yubico



series. If you don't need the other features or form factors and just want to make use of WebAuthn web authentication, this model is a great place to start.

What's WebAuthn All About?

Two-factor authentication (2FA) is something everyone likely is becoming increasingly familiar with. In security, authentication requires one or more of the following:

1. Something you know, like a password.
2. Something you have, like a cryptographic token or officially issued ID card.
3. Something you are, like a person with a specific fingerprint or retinal pattern.

The user name and password combination is the “something you know” type of authentication. Many websites and services now provide users with better security through a second factor. This is typically a one-time password service via Short Message Service (SMS), single-use Time-Based One-Time Password (TOTP) tokens through Google Authenticator or similar applications, push-based TOTP tokens from Duo or LastPass integration, or U2F challenge/response authentication.

U2F is most notably used by Google services, and until quite recently, it has been limited to users with a recent Chrome browser. From a security perspective, U2F and other 2FA techniques are useful tools for increasing web application security. However, there's a growing need for a more seamless user experience, and an ever-increasing need for more robust authentication protocols. As part of an evolving W3C draft standard, WebAuthn provides a better user experience (and arguably a stronger approach to identification and authorization) when using compliant authenticators, such as the YubiKey token.

WebAuthn is designed to be backward-compatible with devices built for the earlier FIDO U2F standard. This includes hardware tokens such as the YubiKey 4, Google's Titan key and various U2F-only devices. However, unlike the YubiKey 5 series, many U2F devices are limited to providing only a hardware-based second factor and aren't

Google Authenticator and Alternatives

Although proprietary systems, such as RSA's SecurID solutions, have been around a long time, the past decade has seen the rise of many alternatives for second-factor authentication. Today's users are likely familiar with SMS solutions that send a six-digit code to their cell phone, but other types of client-side software and hardware tokens have been increasingly on the rise since 2003.

Google Authenticator, initially released in 2010, is arguably the most well known software token solution for typical web users. Google Authenticator leverages a number of open (if not necessarily open-source) protocols, including the widely used QR barcode symbology. Scanning the QR code imports a software token into the user's smartphone, populating a client-side code generator on the device.

Common alternatives to Google Authenticator include:

- FreeOTP (notable for being open-sourced by Red Hat and available under an Apache 2.0 license on GitHub).
- Authy (notable for providing multi-device capabilities).
- LastPass Authenticator.
- Microsoft Authenticator.
- Duo Mobile.

There are certainly many, many others, each with its own set of pros, cons and specialized use cases. This relatively small handful represents the bulk of mind-share among typical users, but you can definitely consider additional alternatives based on issues of cost, open standards support and security track record.

designed for the full suite of multi-factor capabilities provided for by WebAuthn.

As an API, WebAuthn is fundamentally a set of protocols that interacts with CTAP-enabled devices like the YubiKey 5 to provide a comprehensive suite of authentication services that rely on public key cryptography. Depending on the capabilities of both the server and the user's device, WebAuthn supports the following:

- **Single-factor authentication:** passwordless logins, where the presence of the hardware token is sufficient. This is somewhat similar to passwordless SSH keys, only much more secure. A physical YubiKey token won't expose secret key material to the local or remote host systems even when plugged in, and it can't be used at all when physically removed.
- **Two-factor authentication (2FA):** this is typically a user name and password combination (something you know), followed by detection of the hardware token (something you have). This type of 2FA is generally considered more secure than second-factor authentication systems, such as SMS, TOTP or HOTP generators, because of the requirement for a physical token.
- **Multi-factor authentication (MFA):** true MFA often involves a PIN or biometric signature that isn't transmitted over the network. This third factor is typically used locally to unlock the functionality of the hardware token, and it adds another "something you know" or "something you are" layer for successful authentication.

WebAuthn, Overly Simplified

The current working draft of the WebAuthn specification is more than 100 pages long, so describing it in simple terms runs the risk of over-simplification. With that in mind, this section is not a comprehensive guide to the entire WebAuthn API. It's intended to be a *useful* abstraction that highlights how WebAuthn works behind the scenes, highlighting the value of the YubiKey in simplifying the web registration and authentication process.

The heart of WebAuthn is the challenge/response that takes place between a “relying party server” (aka the remote service, such as a website) and a token in the user’s possession. The server issues a challenge that is ultimately received by your browser, which then interacts with the YubiKey or another U2F- or FIDO2-compliant token. This token is called the “authenticator”.

With a hardware token like the YubiKey, the authenticator can create a signed response to the server’s challenge without ever exposing the secret key portion of the credentials stored within the token. The server then validates the response from the authenticator to complete the registration or authentication process.

Even with software tokens, as opposed to hardware tokens like the YubiKey, the API is designed to limit the amount of data exposed to the server during token registration or authentication. Although hardware tokens are considered more secure, the WebAuthn API allows for other types of authenticators as well.

With WebAuthn, the user experience involves just a few simple steps:

1. Receive an authentication prompt from your browser.
2. Use your YubiKey to provide a response.
3. Wait a few milliseconds for the WebAuthn framework to validate your YubiKey’s response.
4. Do fun stuff as an authenticated user!

Under the hood though, WebAuthn and the YubiKey are doing a lot more work. Let’s take a closer look.

The Not-So-Simple WebAuthn API

Although the user experience is straightforward, the implementation details are anything but. The WebAuthn API covers two closely related key-management activities

for handling registration and authentication, and it calls these activities “ceremonies”.

Both ceremonies require a Relying-Party Server, a Relying-Party JavaScript Application, a supported web browser and an Authenticator with certain properties.

Currently, supported web browsers include:

- Mozilla Firefox 60+.
- Google Chrome 67+.
- Google Chrome for Android Beta 70+.

Support by Microsoft Edge was introduced into development builds in July 2018.

Meanwhile, Apple is a participant in the WebAuthn working group, but there is no indication of if (or when) the Safari browser will support the WebAuthn API.

Authenticator properties include key management capabilities and the ability to generate cryptographic signatures. FIDO2 authenticators must also be able to map credentials to each 64-byte user handle associated with a given Relying Party, although the underlying implementation details may vary. Since the YubiKey 5 documentation states that it supports “unlimited” credentials, I assume that the mapping is derived from input during the WebAuthn ceremonies, rather than using fixed storage space within the YubiKey token.

Protocol and implementation details related to how the Authenticator Attestation Globally Unique Identifier (AAGUID) is tied to individual tokens without creating privacy concerns, and how attestation of authenticator provenance is handled by X.509 certificates, are certainly important for the security of the WebAuthn system. But while these concerns are addressed within the specification, the average user doesn’t actually need that level of technical detail to use a YubiKey 5 device safely. The elegance of the WebAuthn/YubiKey solution is that the API handles those details for you.

Registration Using the WebAuthn API Before you can use a YubiKey to authenticate to a web service, a U2F- or FIDO2-capable device must *register* with the relying party server through a registration ceremony. There are some potential use cases where registration is optional or when first-time registration is combined with authentication for new enrollments. However, a typical website likely will want to associate a given credential with a *user handle*, which is a 64-byte identifier for a user account.

At the time of this writing, the registration ceremony defined by the API follows seven steps (Figure 4):

1. *Registration request*: the server-side application initiates a registration request. How this is done is not specified by the WebAuthn API. This is currently an application-specific implementation detail.
2. *Server passes input to client-side JavaScript*: the server sends a challenge, along with user information and relying-party data, to a JavaScript application running in the client’s browser.
3. *Browser requests credential from authenticator*: the user’s browser provides sufficient information to an authenticator so the device can generate a unique credential. The YubiKey 5 includes the server-generated user handle when generating a new credential, while older YubiKeys or other U2F-only devices will create a credential with **userHandle** set to null for backward-compatibility.
4. *Authenticator creates an attestation*: the YubiKey creates a cryptographic key pair for the credential and bundles the public key inside a special message called an “attestation statement”.
5. *Authenticator sends attestation object to browser*: the YubiKey signs the attestation statement with a verifiable digital signature and passes the statement along with other data back to the browser as an “attestationObject”.

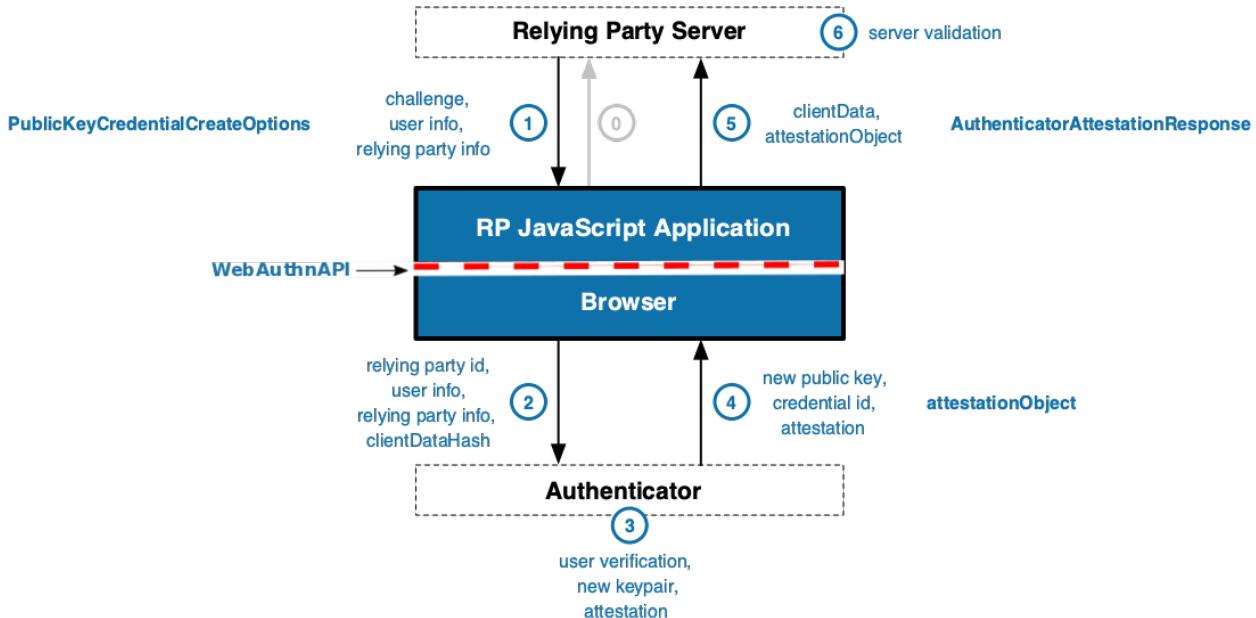


Figure 4. WebAuthn Registration Flow (“Web Authentication: An API for accessing Public Key Credentials Level 1.” W3C Candidate Recommendation, 7 August 2018. <https://www.w3.org/TR/webauthn/#web-authentication-api>)

6. *Credential passed to relying party:* the JavaScript application packs up the attestationObject, along with some JSON and encoded data, and sends it back to the relying party Server as an “AuthenticatorAttestationResponse”.
7. *Server validates response:* the server then validates the response and the credential’s digital signature. If all the validations succeed, the server completes the registration process by associating the public key in the attestation with the user’s account. This public key can then be used for immediate or future authentication.

To be quite frank, even this “deep dive” into the registration process glosses over a great deal of cryptography and message-passing. However, the beauty of WebAuthn, especially when paired with a YubiKey, is that all of this cryptographic work and interprocess communication is essentially invisible to end users. In actual use, basic registration simply involves inserting a YubiKey in response to a browser prompt and

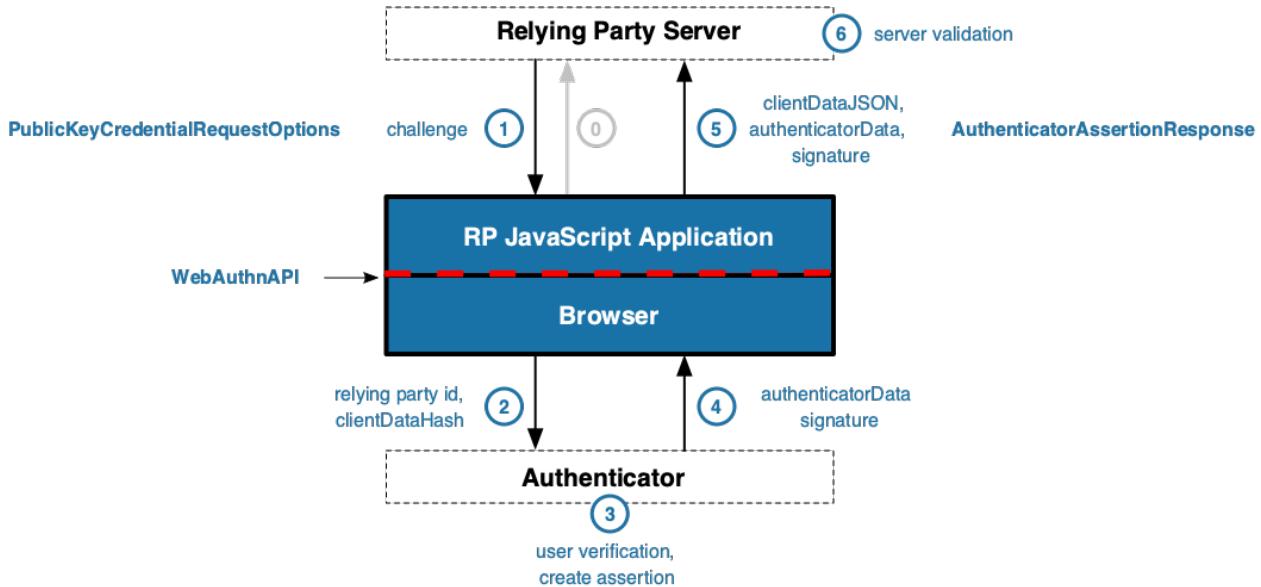


Figure 5. WebAuthn Authentication Flow (“Web Authentication: An API for accessing Public Key Credentials Level 1.” W3C Candidate Recommendation, 7 August 2018. <https://www.w3.org/TR/webauthn/#web-authentication-api>)

tapping the touch-sensitive part of the YubiKey (which varies by model) to activate it.

Once the authenticator is registered, you’re only halfway done. You’ve registered a WebAuthn credential using a YubiKey, but you still have to present the newly registered credentials to the website before you’re actually authorized. In practice, websites can present registration and authentication as a seemingly unified process from the user’s point of view, but they’re actually different ceremonies within the WebAuthn API.

Authentication with the WebAuthn API The process for authentication is actually quite similar in its outlines to the registration process. Although protocols and messages may vary, the key difference is that the relying party server and the authenticator are validating an *existing* credential that was created during the registration process described above.

As with registration, the user perspective is simply to insert the YubiKey and trigger it with a touch. What could be easier?

Passwordless Login Example

If you have a YubiKey device that supports U2F or FIDO2, you can test out registration and authentication using a number of publicly available testing services. For this set of examples, let's use the site provided by Duo Security for

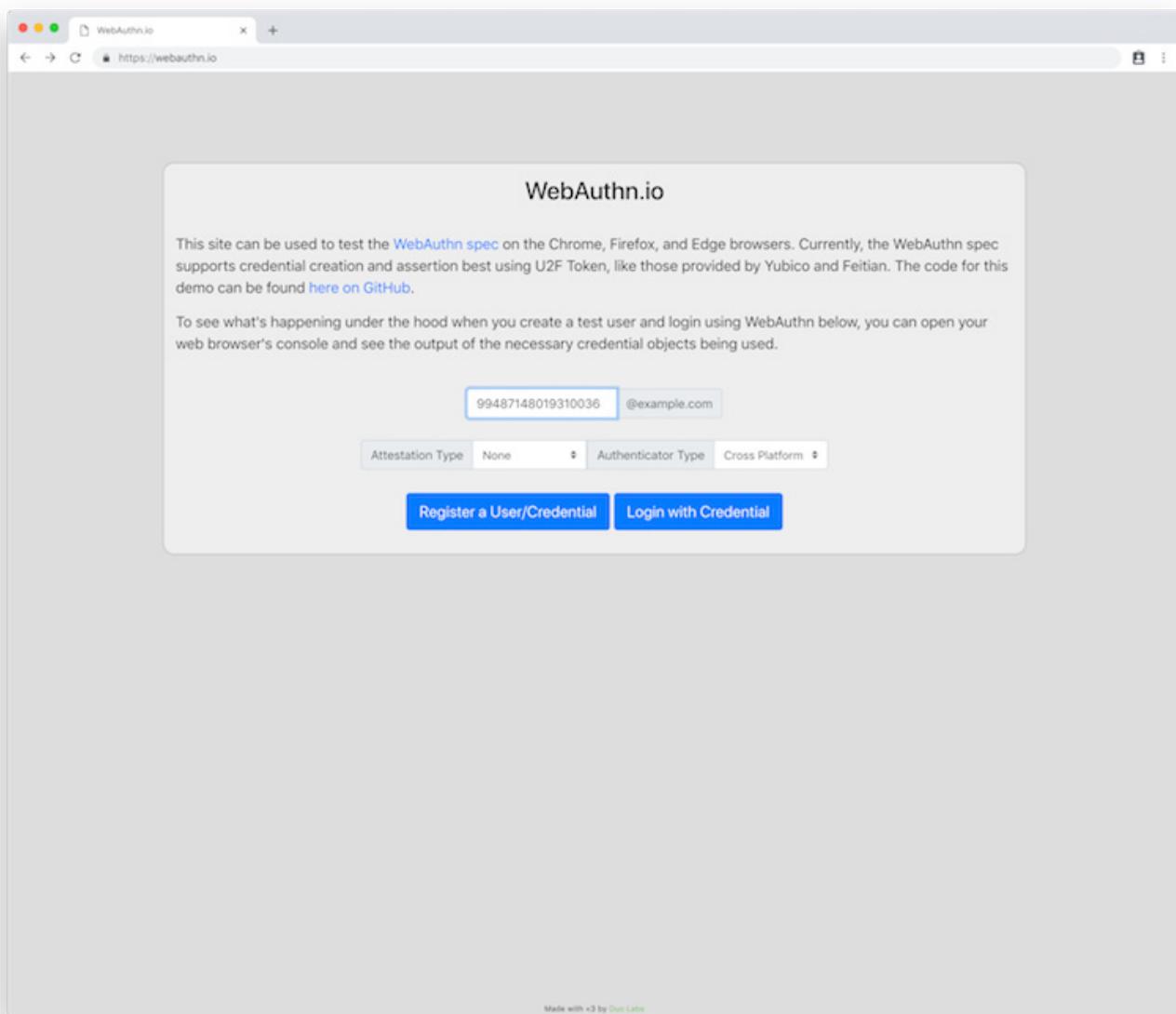


Figure 6. Register Credential

YubiKey testing.

Launch your browser, and navigate to <https://webauthn.io>. Next, register your YubiKey by entering a unique user name. As many other users are likely testing their keys too, I recommend using a UUID for your user name. Running the **uuidgen** command, available on most Linux and macOS systems, will print a value that is sufficiently random for this purpose.

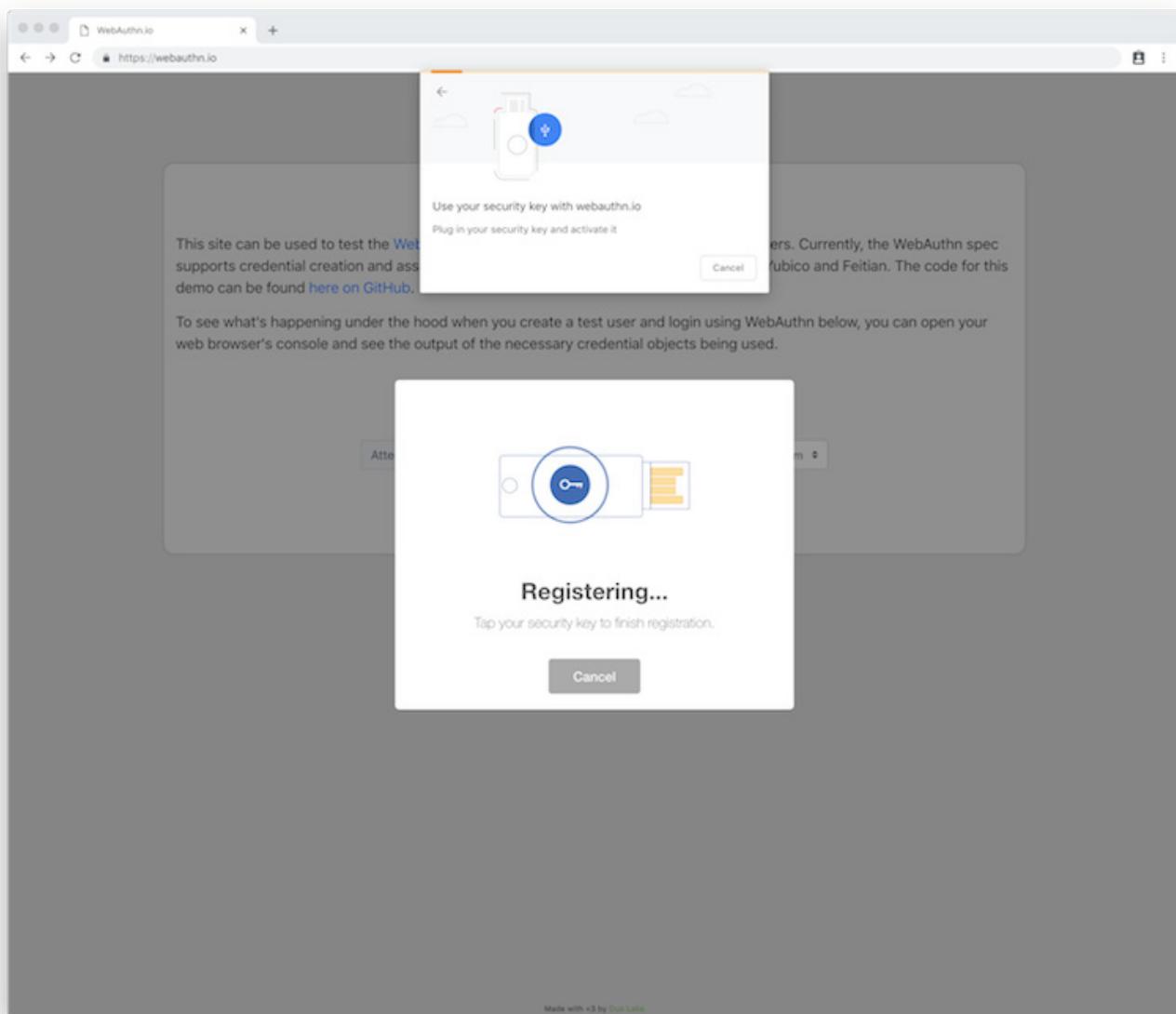


Figure 7. Tap YubiKey

DEEP DIVE

For this exercise, leave the other values, such as Attestation Type and Authenticator Type, at their defaults.

After you've entered your user name, click Register a User/Credential. You'll then be prompted to tap your YubiKey to complete the registration process.

Note that this particular application shows two different dialog boxes. Other

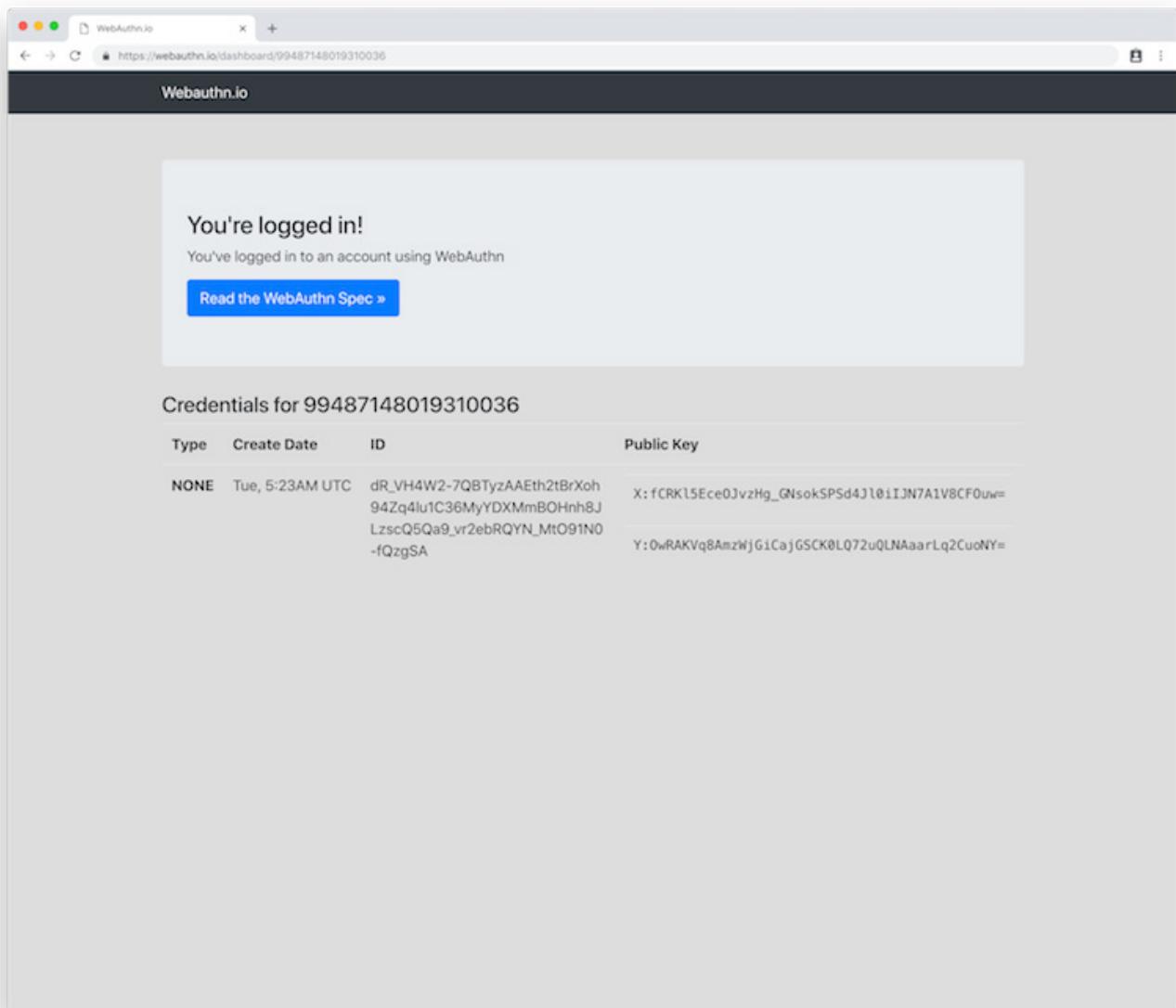


Figure 8. Registered and Logged In

WebAuthn sites show only the top modal dialog; the second (and admittedly prettier) dialog seems to be unique to the Duo website.

In any case, once you've tapped your YubiKey, you'll be registered and logged in immediately. You'll see a screen similar to the one shown in Figure 8.

If you return to the home page, you'll again see the login screen. To test that your YubiKey is working properly, enter your user name and then click Login with Credential. No password required!

Although this example seems trivial, it highlights how easy it is for a user to register or authenticate with a YubiKey. Even though the underlying implementation is complex, the user experience is smooth and simple.

Privacy Considerations

When properly implemented, WebAuthn and compliant tokens like the YubiKey robustly guard user privacy while providing strong user authentication features. Sections of the standard address ways to prevent data correlation, de-anonymization and the use of credentials without user consent.

A full examination of WebAuthn privacy issues is outside the scope of this article, but section 14 of the standard is titled “Privacy Considerations”. If you’re an IT auditor, security administrator, security engineer or application programmer, section 14 will be useful to you.

Cryptographic Weaknesses

In late August 2018, security researchers from Paragon Initiative Enterprises raised concerns about the WebAuthn API’s use of certain algorithms, specifically Elliptic Curve Direct Anonymous Attestation (ECDAA) and RSA with PKCS1v1.5 padding. ZDNet picked up the story and added some additional clarification.

The short version for non-cryptographers is that certain algorithms in the standard represent *potential* weaknesses. These weaknesses are not currently

easy to exploit in practice, but they may present problems for the future. The researchers' cautions are aimed more at API implementers than users, with the goal of improving the WebAuthn standards before they're finalized. The researchers themselves say:

WebAuthn and ECDAA are not doomed. Don't throw away your hardware tokens, revert your codebases to use SMS or TOTP, or any other such drastic measures. ■



Todd A. Jacobs is the CEO of Flow Capital Group, which acquires and manages companies specializing in IT automation, DevOps & agile transformations, security and compliance, fractional CIO/CTO services, and board advisory services for cyber risk and other hot technology issues. Todd waited his whole life for Matt Smith's character on *Dr. Who* to make bow ties cool again. He lives near Baltimore, MD with his wife and son, to whom he hopes to pass on his love of Linux and technology—but perhaps not his fashion sense.

Send comments or feedback
via <https://www.linuxjournal.com/contact>
or email ljeditor@linuxjournal.com.

Resources

- “Web Authentication: An API for accessing Public Key Credentials Level 1.” W3C Candidate Recommendation, 7 August 2018
- FIDO 2.0: Client To Authenticator Protocol
- FreeOTP Two-Factor Authentication
- Server-side Web Authentication library for Java
- WebAuthn Rails Demo App
- Ruby implementation of a WebAuthn Relying Party
- WebAuthn/FIDO2 JavaScript application
- Mozilla Web Authentication API
- “Introducing Web Authentication in Microsoft Edge”
- “Security Concerns Surrounding WebAuthn: Don’t Implement ECDAA (Yet)”
- “Worries arise about security of new WebAuthn protocol” (*ZDNet*)
- FIDO ECDAA Algorithm
- WebAuthn Privacy Considerations

The Purism Librem Key

The Librem Key is a new hardware token for improving Linux security by adding a physical authentication factor to booting, login and disk decryption on supported systems. It also has some features that make it a good general-purpose OpenPGP smart card. This article looks at how the Librem Key stacks up against other multi-factor tokens like the YubiKey 5 and also considers what makes the Librem Key a unique trusted-computing tool.

By Todd A. Jacobs

Purism is a new player in the security key and multi-factor authentication markets. With the introduction of the Librem Key, Purism joins the ranks of other players—such as Yubico, Google, RSA and so on—in providing hardware tokens for multi-factor authentication.

In addition, like the YubiKey 5 series, the Librem Key also provides OpenPGP support with cryptographic functions that take place securely on-key. This allows users to generate and use GnuPG public and private keys without exposing any secret key material to the host computer where the USB device is attached.

The Librem Key is based on the German-manufactured Nitrokey Pro 2, but it has been modified to focus on “trusted boot” when used with Purism’s Linux laptops. (I take a closer look at what the trusted boot process is and how the Librem Key fits into that process, later in this article.)

Comparing the Librem Key to the YubiKey 5

There is certainly overlap between the features of the Librem Key and the YubiKey 5 series. Let’s look at what they have in common before I go into what makes the Librem Key unique.

*DEEP
DIVE*

Table 1. Librem Key and YubiKey Feature Comparison

FEATURE	Librem Key	YubiKey 5
OpenPGP support	yes	yes
PAM support	yes	yes
PIV smart card	no	yes
HOTP support	yes	yes
TOTP support	yes	yes
Password management	yes	yes
PKCS#11 support	yes	yes
S/MIME support	yes	yes
X.509 support	yes	yes
FIDO U2F	no	yes
FIDO2	no	yes
Hardware TRNG	yes	no
USB-A	yes	yes
USB-C	no	yes

As you can see from Table 1, the two devices are more alike than they are different. Both devices can be used for the following:

1. PAM-enabled logins on Linux systems.
2. One-time password credentials, such as TOTP and HOTP.

3. OpenPGP support, including onboard key generation and slots for encryption, signing and authentication keys.
4. Password management: two configurable slots on the YubiKey with touch-to-send and touch-to-generate functionality once the device is configured, and 16 entries on the Librem Key when used with the Nitrokey App supporting application.
5. Source of randomness using the OpenSC protocol: hardware randomness from the Librem Key and algorithmic randomness from the YubiKey.



Figure 1. Purism Librem Key (Photo Credit: Purism, SPC)



**Figure 2. Nitrokey Pro (Photo
Credit: Nitrokey UG)**

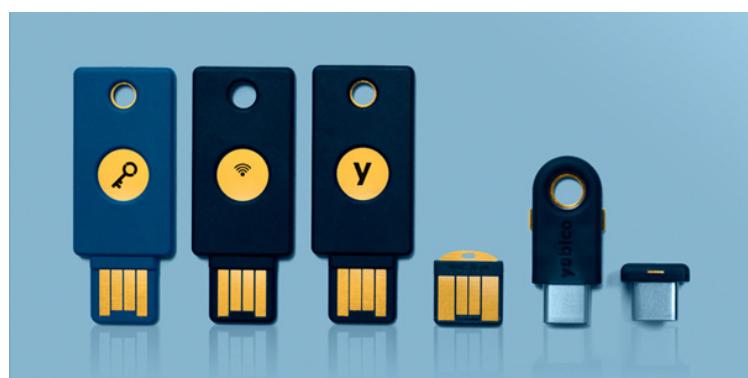


Figure 3. YubiKey 5 Series Form Factors

There also are areas where the YubiKey 5 series and certain Nitrokey models offer more features than the Librem Key. In particular, the YubiKey comes in more form factors, and it's significantly thinner or smaller than the chunkier thumb-drive form factor of the Librem Key.

The YubiKey offers 32 slots for onboard HMAC-based One-Time Password (HOTP) storage. In contrast, the Librem Key currently holds only three.

The Nitrokey offers FIDO U2F and secure on-key storage on several of its models, although you can't get both features on the same device. Since the Nitrokey Pro 2 doesn't offer either of these features currently, the Librem Key doesn't either.

Of the three devices, the YubiKey offers the most options for multi-factor authentication. In direct comparison, the Librem Key supports far fewer slots and protocols.

However, feature-by-feature and form-factor comparisons don't do the Librem Key justice. The unique value of the device is in its approach to improving Linux system security. Let's take a closer look at how that's done.

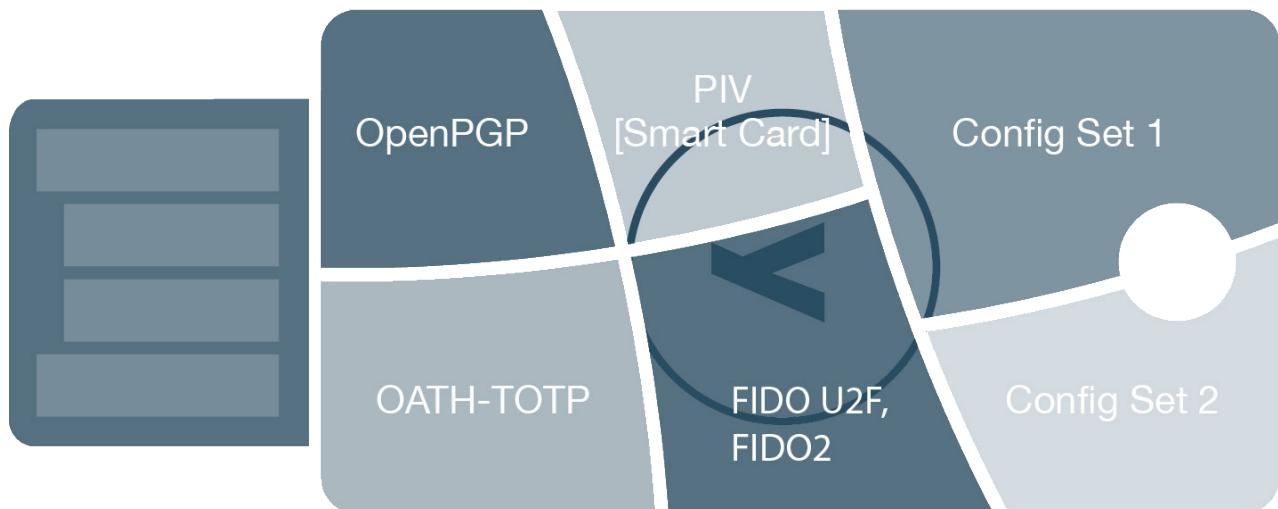


Figure 4. YubiKey Diagram (Photo Credit: Yubico, Inc.)

Librem Key for Secure Boot

Purism currently sells a set of Linux laptops aimed at security-conscious users. These laptops include unique features like a physical toggle for the camera and microphone, and a second switch for toggling WiFi and Bluetooth.

The laptops come with PureOS installed, as well as a Heads-enabled coreboot process that leverages each system's Trusted Platform Module (TPM). When paired with the Librem Key, these components collectively verify the integrity of the boot process. Purism says:

[T]he Librem Key makes it easy to prove your system is secure by detecting whether your laptop BIOS or kernel has been tampered with. Insert the key at boot time and if it blinks green, all systems are go. If it ever detects tampering, the Librem Key's LED blinks red, alerting you to the problem.

Note: see *Kyle Rankin's "Tamper-Evident Boot with Heads" article in this issue* for a more indepth look at the Heads project.



Figure 5. Librem Hardware Switches (Photo Credit: Purism, SPC)

Heads—which is probably a play on the name of the security-focused distribution Tails—displays a time-based one-time password (TOTP) at boot calculated from various BIOS measurements. You then can verify this one-time code against the code generated in a multi-factor app, such as Google Authenticator or FreeOTP. Matching codes provide cryptographic guarantees that the boot process has not been tampered with.

In this scenario, you aren't actually using the one-time password to authenticate yourself to the system. Instead, the system is authenticating itself to you, the user, so that you can verify its integrity!

With Heads, you can boot the computer without validating the TOTP. This ensures that the validation process won't block you if you don't have access to your smartphone or other validation device, but doing so is clearly a security trade-off that swaps strong verification for convenience at your discretion.

Removing the Librem Key's reliance on the host computer's system clock increases security further, since the Librem Key doesn't have its own independent time source. Purism modified the Nitrokey Pro 2 to use HOTP instead of TOTP. This functionality is paired with a boot-time application that attempts to communicate with a connected Librem Key to validate the HOTP code in a more visual and automated way.

If a Librem Key is detected, and everything is as it should be, the key flashes green as a visual indication that your TPM and BIOS haven't been tampered with. If it flashes red, your system's boot process has failed the integrity check, and it may have been tampered with. Depending on the type of tampering, the Heads system (which may itself have been tampered with) *should* report an error, but the tamper-resistant Librem Key will reliably display a flashing red light to warn you of an unsafe system.

As a thoughtful design choice, a lost Librem Key won't lock you out of your system. Although you temporarily will lose the visual indicator and automated self-checking mechanism, you still can use the boot system's standard TOTP mechanism for manual verification until you find or replace your Librem Key.

Not Just for Purism Laptops

Although this article focuses on using the Librem Key with Purism laptops, it's worth noting that the underlying magic is the combination of four key components:

1. A computer system with a TPM module.
2. The Heads coreboot system.
3. The `libremkey_hotp_initialize` command-line tool.
4. The `libremkey_hotp_verification` command-line tool.

In theory, anyone running PureOS on a TPM-enabled system should be able to make use of the Librem Key's secure-boot functionality. Additionally, any distribution that can use Heads modified with the command-line tools listed above also could take advantage of a Librem Key.

Although a Librem Key is obviously most useful out of the box when paired with a Purism laptop, the open-source nature of the solution makes it an ideal playground for Linux enthusiasts on other TPM-enabled hardware as well. Purism deserves credit for providing an open-source security solution without vendor lock-in!

Librem Key for Disk Decryption

The Librem Key also is intended to support automatic decryption of LUKS-encrypted disks simply by having the key inserted at boot time. However, this functionality currently is pending support from upstream Debian maintainers, followed by some additional work by the PureOS team. The problem is likely to be resolved by the time this article is published, but it remains outstanding at the time of this writing.

While LUKS decryption with the Librem Key is not yet available, the device could be used by other tools such as VeraCrypt to provide keyfile-based decryption or other workarounds.

The Librem Key provides PKCS#11 support. That means it should be compatible with VeraCrypt's smart card and hardware token support. VeraCrypt allows smart card tokens to be used as keyfiles for cryptographic operations such as disk decryption. The following comes directly from VeraCrypt's keyfile documentation:

VeraCrypt can directly use keyfiles stored on a security token or smart card that complies with the PKCS #11 (2.0 or later) standard and that allows the user to store a file (data object) on the token/card. To use such files as VeraCrypt keyfiles, click Add Token Files (in the keyfile dialog window).

Access to a keyfile stored on a security token or smart card is typically protected by PIN codes, which can be entered either using a hardware PIN pad or via the VeraCrypt GUI. It can also be protected by other means, such as fingerprint readers.

In order to allow VeraCrypt to access a security token or smart card, you need to install a PKCS #11 (2.0 or later) software library for the token or smart card first. Such a library may be supplied with the device or it may be available for download from the website of the vendor or other third parties.

If your security token or smart card does not contain any file (data object) that you could use as a VeraCrypt keyfile, you can use VeraCrypt to import any file to the token or smart card (if it is supported by the device). To do so, follow these steps:

- 1) In the keyfile dialog window, click Add Token Files.
- 2) If the token or smart card is protected by a PIN, password or other means (such as a fingerprint reader), authenticate yourself (for example, by entering the PIN using a hardware PIN pad).

- 3) The ‘Security Token Keyfile’ dialog window should appear. In it, click Import Keyfile to Token and then select the file you want to import to the token or smart card.

While this workaround *should* work, there’s an important caveat. Because of initial high demand for the Purism Librem Key, I was unable to get a hold of a key to test this configuration. This alternative approach is offered in the spirit of can-do Linux hacking rather than as a tried-and-true method. Your mileage may therefore vary.

As yet another option, the ability to use the Librem Key to encrypt or decrypt documents, passwords (such as with the pass command-line password manager) or block devices using standard OpenPGP operations works as expected.

Finally, it’s worth noting that existing LUKS, dm-crypt and ecryptfs options are typically “good enough” for most users, provided that you can trust the integrity of your BIOS and operating system. Since PureOS and the Librem Key already offer those integrity guarantees on TPM-enabled systems, the main benefit of using the Librem Key to unlock encrypted disks is the tamper-resistant *automation* of the decryption process.

Conclusion

If you want to implement trusted boot, or prefer to work with a fully open-source OpenPGP smart card, the Librem Key is a great choice. For other uses, the value proposition is less clear.

The Librem Key is an early-stage product. This is most apparent in its smaller feature set and less-rugged construction when compared to the YubiKey.

At a retail price of \$59 plus shipping, the Librem Key is also pricier than comparable products from Yubico. It’s also slightly more expensive than the original Nitrokey Pro 2 on which it’s based.

As an OpenPGP smart card or hardware-based authentication token, the Librem Key falls a little short of its competition in features and pricing. However, its use of open-

Glossary

- **FIDO U2F:** FIDO Universal 2nd Factor Authentication.
- **FIDO2:** a second-generation U2F protocol that is part of the W3C WebAuthn framework.
- **HOTP:** HMAC-based One-time Password algorithm defined by RFC 4226 and frequently used by OAuth-enabled systems.
- **LUKS:** Linux Unified Key Setup.
- **OATH:** Initiative for Open Authentication.
- **OTP:** One-Time Password.
- **PIV:** Personal Identity Verification, usually associated with the FIPS 201 standards.
- **PRNG:** Pseudo-random number generators approximate randomness by applying software algorithms to a seed value. Contrast with TRNG.
- **TOTP:** Time-based One-Time Password algorithm.
- **TRNG:** a true (or hardware) random number generator generates the random values important to cryptography through physical processes. Contrast with PRNG.

source hardware, firmware and software make the Librem Key a very compelling alternative to similar tokens with proprietary firmware (such as the YubiKey) when optimizing for trust and transparency.

DEEP DIVE

Furthermore, no other consumer product on the market currently provides hardware validation of a trusted boot process the way Purism's Librem Key does. If hardware tamper-detection is important to you, the use of a Librem Key with compatible hardware and software should be an integral component of your defense-in-depth security strategy.

A fully trusted boot process is essential to effective computer security. The Librem Key increases security by validating that process each and every time. The visual green/red indicator automates the validation process and greatly simplifies the user experience.

The Librem Key is a unique product and a significant step forward in user-friendly trusted-boot authentication. For its intended use case, it's the best option on the market today—and a product worth following as it continues to evolve. ■



Todd A. Jacobs is the CEO of Flow Capital Group, which acquires and manages companies specializing in IT automation, DevOps & agile transformations, security and compliance, fractional CIO/CTO services, and board advisory services for cyber risk and other hot technology issues. Todd waited his whole life for Matt Smith's character on *Dr. Who* to make bow ties cool again. He lives near Baltimore, MD with his wife and son, to whom he hopes to pass on his love of Linux and technology—but perhaps not his fashion sense.

Send comments or feedback
via <http://www.linuxjournal.com/contact>
or email ljeditor@linuxjournal.com.

Resources

Software and Documentation:

- [coreboot](#)
- [dm-crypt](#)
- [ecryptfs](#)
- [Heads](#)
- [Librem Key Specifications](#)
- [Librem Key Tamper Detection](#)
- [Nitrokey](#)
- [Nitrokey HOTP Verification](#)
- [pass Password Manager](#)
- [PureOS](#)
- [VeraCrypt](#)
- [VeraCrypt Keyfiles](#)
- [YubiKey 5 NFC specifications](#)

Bugs and Tickets:

- [Debian bug: OpenPGP support for unlocking encrypted volumes](#)
- [PureOS ticket: OpenPGP smartcard and LUKS integration](#)

Tamper-Evident Boot with Heads

Learn about how the cutting-edge, free software Heads project detects BIOS and kernel tampering, all with keys under your control.

By **Kyle Rankin**

Disclaimer: I work for Purism, and my experience with Heads began as part of supporting it on Purism's hardware. As a technical writer, I personally find ads that mask themselves as articles in technical publications disingenuous, and this article *in no way* is intended to be an advertisement for my employer. However, in writing this deep dive piece, I found that mentioning Purism was unavoidable in some places without leaving out important information about Heads—in particular, the list of overall supported hardware and an explanation of Heads' HOTP alternative to TOTP authentication, because it requires a specific piece Purism hardware.

Some of the earliest computer viruses attacked the boot sector—that bit of code at the beginning of the hard drive in the Master Boot Record that allowed you to boot into your operating system. The reasons for this have to do with stealth and persistence. Viruses on the filesystem itself would be erased if users re-installed their operating systems, but if they didn't erase the boot sector as part of the re-install process, boot sector viruses could stick around and re-infect the operating system.

Antivirus software vendors ultimately added the ability to scan the boot sector for known viruses, so the problem was solved, right? Unfortunately, as computers, operating systems and BIOSes became more sophisticated, so did the boot-sector attacks. Modern attacks take over before the OS is launched and infect the OS itself, so when you try to search for the attack through the OS, the OS tells you

everything is okay.

That's not to say modern defenses to this type of attack don't exist. Most modern approaches involve proprietary software that locks down the system so that it can boot only code that's signed by a vendor (typically Microsoft, Apple, Google or one of their approved third-party vendors). The downside, besides the proprietary nature of this defense, is that you are beholden to the vendor to bless whatever code you want to run, or else you have to disable this security feature completely (if you can).

Fortunately, an alternative exists that is not only free software, but that also takes a completely different approach to boot security by alerting you to tampering instead of blocking untrusted code. This approach, Heads, can detect tampering not only in the BIOS itself but also in all of your important boot files in the /boot directory, including the kernel, initrd and even your grub config. The result is a trusted boot environment with keys fully under your own control.

In this article, I describe some of the existing boot security approaches in more detail, along with some of their limitations, and then I describe how Heads works, and how to build and install it on your own system.

Why Boot Security Matters

To understand why having a secure boot process matters so much, it's useful to understand one of the most common threats on a Linux system: rootkits. A rootkit is a piece of software attackers can use to exploit vulnerabilities in the kernel or other software on the system that has root privileges, so it can turn normal user-level access into root-level access. This ability to escalate to root privileges is important, because although in the old days, all network services ran as root, these days, servers more often run as regular users. If attackers find a flaw in a network service and exploit it so they are able to run commands locally, they will only be able to run those commands as the same user. The rootkit allows them to turn those local user privileges into root privileges, whereby they then can move on to the next step, which is installing backdoors into your system, so they can get back in later undetected.

Although sometimes attackers will install a backdoor that just has a service listening on an obscure port all the time, kernel backdoors are preferred because once they exploit the kernel, they then can mask any attempts by your OS to detect the attack. After all, if you want to know what files are in a directory, or which processes are running, you have to ask the kernel. If you can exploit the kernel, you can hide your malicious processes or files from prying eyes. Many rootkits also will set up a kernel backdoor for attackers automatically as part of the automated attack.

Rootkits aren't only a threat on servers; it's just that servers are accessible on the network all the time, and they run software that listens for requests. Although modern Linux desktop installs don't have any services listening on the network, there still are plenty of ways for attackers to launch code locally as your user—via the web browser is one of the most common ways, and malicious file attachments in email is another.

The whole point of a rootkit is to make it difficult for you to detect it from the running OS, but you still always can boot the system from a live USB-based OS and examine the hard drive. Or, you could re-install the OS completely and be rid of the threat. Yet even in that case, you are relying on the BIOS to boot your live USB-based OS. Your BIOS is the first code your CPU executes when it boots. Once it loads, it detects the hardware on your system, initializes it, and then lets you boot either from an internal hard drive or perhaps from external USB or DVD media. If attackers were able to modify your BIOS, in theory, they could just re-install their backdoor in any kernel it loads and persist even with re-installing the OS or examining it from a live USB disk.

The BIOS then becomes the root of trust for the entire rest of the system. Until you can trust it, you can't fully trust the rest of the code that executes after it.

Next I describe some of the current approaches to secure the boot process, all of which involve executing only pre-approved code.

Other Boot Security Methods

It's easier to understand how Heads works, and how it is different from the existing

approaches, once you understand how the existing approaches work. The main two approaches that provide boot security on modern systems are UEFI Secure Boot and Intel Trusted Boot.

UEFI Secure Boot Of all of the different approaches to secure the boot process, UEFI Secure Boot is the most popular, and it's included in just about every modern laptop and desktop you would buy. The way that Secure Boot works is that the UEFI flash chip contains certificates for Microsoft and its approved third-party vendors. UEFI boot firmware that works with Secure Boot contains a signature created by the private keys of either Microsoft or its approved vendors. Secure Boot then checks that signature against its certificates, and if the signature matches, it allows the boot firmware to execute. If the signature doesn't match or is missing, Secure Boot will not allow it to run.

Because it was initially designed for Windows, and initially Windows was the only OS that used it, Secure Boot often is thought of as a Microsoft-only technology, and many in the FOSS community spoke out against it because of the risk that it could be used to lock out a system from loading Linux. It's true that initially you could use Secure Boot only with Windows, but Linux distribution vendors like Red Hat and Ubuntu worked with Microsoft to get a boot "shim" signed that would allow them to load GRUB and boot their OSes.

Of course, there still are plenty of Linux distributions that haven't gotten boot shim code signed by Microsoft, including Debian. This means that if you want to install Debian on a system with Secure Boot, you first must go into your UEFI settings and disable Secure Boot entirely before you are allowed to boot the USB installer—that is, if your UEFI software allows you to disable Secure Boot. Some lower-cost computers these days ship with stripped-down UEFI firmware that allow only a very minimal level of configuration, and on these systems, Secure Boot often is no longer optional.

Secure Boot does have a mechanism that would allow you to replace the existing vendor certificates with your own, and that might be an option for Linux users who want to use Secure Boot on systems that don't use Microsoft-signed boot firmware.

The process itself is somewhat complicated though, and the end result would boot your own custom-signed code but then would lock out anything not signed with your own signatures, such as a typical USB OS installer. Again, this is an option only if you first can disable Secure Boot to load your untrusted OS and modify UEFI, or else attempt the modification from a trusted OS.

Intel Trusted Boot Along with Secure Boot, modern Intel computers also have the option of a security mechanism called Intel Trusted Boot. This mechanism takes advantage of the special capabilities of the Trusted Platform Module (TPM) chip on a system. The TPM is a standalone chip available on some motherboards that can act as its own Hardware Security Module (HSM) by generating its own cryptographic keys and performing cryptographic operations on-chip independent of the system CPU. The TPM also contains Platform Configuration Registers (PCRs) that can contain measurements of executed code in the form of a chain of hashes. Generally, different PCRs are used to store measurements of different phases of the boot process.

Intel Trusted Boot works by sending the measurements of code as it is executing over to the TPM where it is hashed and stored in a corresponding PCR. As new code is executed, it also gets hashed and combined with hashes of previous code in the PCR. The TPM allows you to seal secrets (disk decryption keys are common) within it that are unlocked only if the PCRs contain previously stored values. Combined with Secure Boot, Intel Trusted Boot allows you to detect tampering in boot-time executables.

Secure Boot Limitations

Secure Boot is the main way vendors provide boot-time security on modern computers, but it has quite a few limitations. The first big limitation is also its biggest claimed feature—that it requires boot code to be signed by keys under the vendor’s control. This means if you did happen to want to run custom boot code, you must work with vendors to get them to sign your binary or else replace all of their certificates with your own and run *only* your own code.

Another limitation is that although Secure Boot ensures that you are running code that has been signed, it doesn’t ensure that you are running the *same* boot code

that you ran previously. An attacker who was able to get access to one of the vendor signing keys could create a boot-time executable that would pass Secure Boot protections. What would happen to existing computers if one of the Microsoft (or other vendor) signing keys were leaked or forced to be shared with a nation state?

Secure Boot is also proprietary software, so you have to take vendors at their word that there are no backdoors within it, and you also have less visibility into what code might be signed. In addition, Secure Boot validates only executables. It can't validate your initrd files or GRUB configs—both places where attackers could add malicious changes. Ultimately, the issue with Secure Boot is that it takes control of your computer and its security out of your hands and into the hands of vendors. If you fully trust your vendor, perhaps you are fine with that trade-off, but many people would prefer to have full control over their own software and hardware.

Introduction to Heads

Heads was created by Trammell Hudson to solve some of the trust issues and other limitations of Secure Boot by replacing it with a system that focuses on detecting tampering instead of blocking it. The idea with Heads is to capture a stable, trusted state in the BIOS and boot code, and then ensure that at each subsequent boot, the BIOS and boot code haven't changed. Heads is written under a free software license, so it not only can be inspected, but it also is reproducibly built, so if you were to get a pre-built Heads ROM, you also could build the same revision of Heads yourself and get the same result, thereby proving that the code wasn't tampered with at some point in the build process.

Heads loads from within the open-source coreboot BIOS (or optionally LinuxBoot for some server platforms) and is actually its own standalone Linux kernel and runtime environment that performs tamper checks and then boots into your system kernel once everything checks out. Unlike with Secure Boot, it detects tampering using keys that are fully under your control—keys you can change at any point.

Hardware Support

Because Heads relies on coreboot or LinuxBoot, its current hardware support is

somewhat limited to hardware that both supports either coreboot or LinuxBoot, has a TPM, and has someone who has defined that board's configuration, including coreboot settings and other options, and submitted it to Heads. Currently, that list is pretty small: Lenovo ThinkPad X220 and X230, the Purism Librem laptop line and a handful of servers.

How Heads Works

On the surface, Heads works similarly to Intel Trusted Boot in that it uses the TPM to verify measurements of itself to then unlock a secret. That's where the similarities end though, as Heads approaches boot security in a much different way, because its aim is to provide tamper *detection*, not tamper *proofing*. Heads will alert you to tampering, but it still provides you the ability to boot whatever software you want.

You can break down the default Heads boot process into a few main phases:

- The coreboot BIOS starts and loads the Heads kernel and initrd.
- As code executes, measurements are sent over to the TPM chip.
- Heads presents a TOTP/HOTP code to prove to the user that it hasn't been tampered with.
- The user selects a boot option.
- Heads checks all the files in /boot for tampering before loading the OS.
- If the files all check out, Heads boots the OS.

Heads uses two different sets of keys to detect tampering. First it uses a shared secret stored in the TPM and also on either a TOTP authenticator application on your phone or on a special USB security token like the Librem Key. This shared secret is used to prove the BIOS itself hasn't been tampered with. The next set of keys is a set of trusted GPG public keys within a GPG keyring that you add to the Heads ROM. Once

you know the BIOS hasn't been tampered with, you can trust that the GPG keyring it has within it hasn't been modified to add an untrusted key. Heads then uses that trusted keyring to verify all of the signatures on the files in /boot. In both cases, these are secrets that are fully under your control, and you can change them and reset signatures at any point.

Next, let's look at more specifics of how Heads works by focusing on each of these two secrets and how they are used in their respective parts of the boot process.

Boot Security and the TPM

The very first thing Heads must do is prove to you that it can be trusted and that it hasn't been tampered with. The challenge is, if it *has* been tampered with, couldn't it lie to you and tell you everything is okay? This is where the TPM comes in. When you first set up Heads, you go through a process to reset the TPM and set up a new admin password (called *taking ownership*), and then Heads will generate a random secret and store it in the TPM (called *sealing*) along with the current valid measurements it will take to unlock that secret.

Once the secret is sealed in the TPM, Heads will convert that secret into a QR code and display it on the screen, so you can scan it with your phone to add it to your TOTP authenticator application of choice (FreeOTP is a free software option that works on Android, for instance). If you have added Librem Key support into Heads, you also can store a copy of the secret onto Purism's Librem Key USB security token.

When Heads boots, it then sends measurements of the code it executes over to the TPM. If the BIOS has been tampered with, those measurements won't match what was there before, and the TPM will not unlock the shared secret. In that case, Heads will output an error to the screen alerting you to the problem. If the measurements do match, the TPM will unlock the shared secret, send it to Heads, and Heads will combine the secret with the current time to convert it to a six-digit TOTP code that it will display on the screen. You then can compare that code to the six-digit TOTP code in your phone's app, and if they match, you know that the secret was valid. Alternatively, if you enabled Librem Key support, you could insert the device at boot,

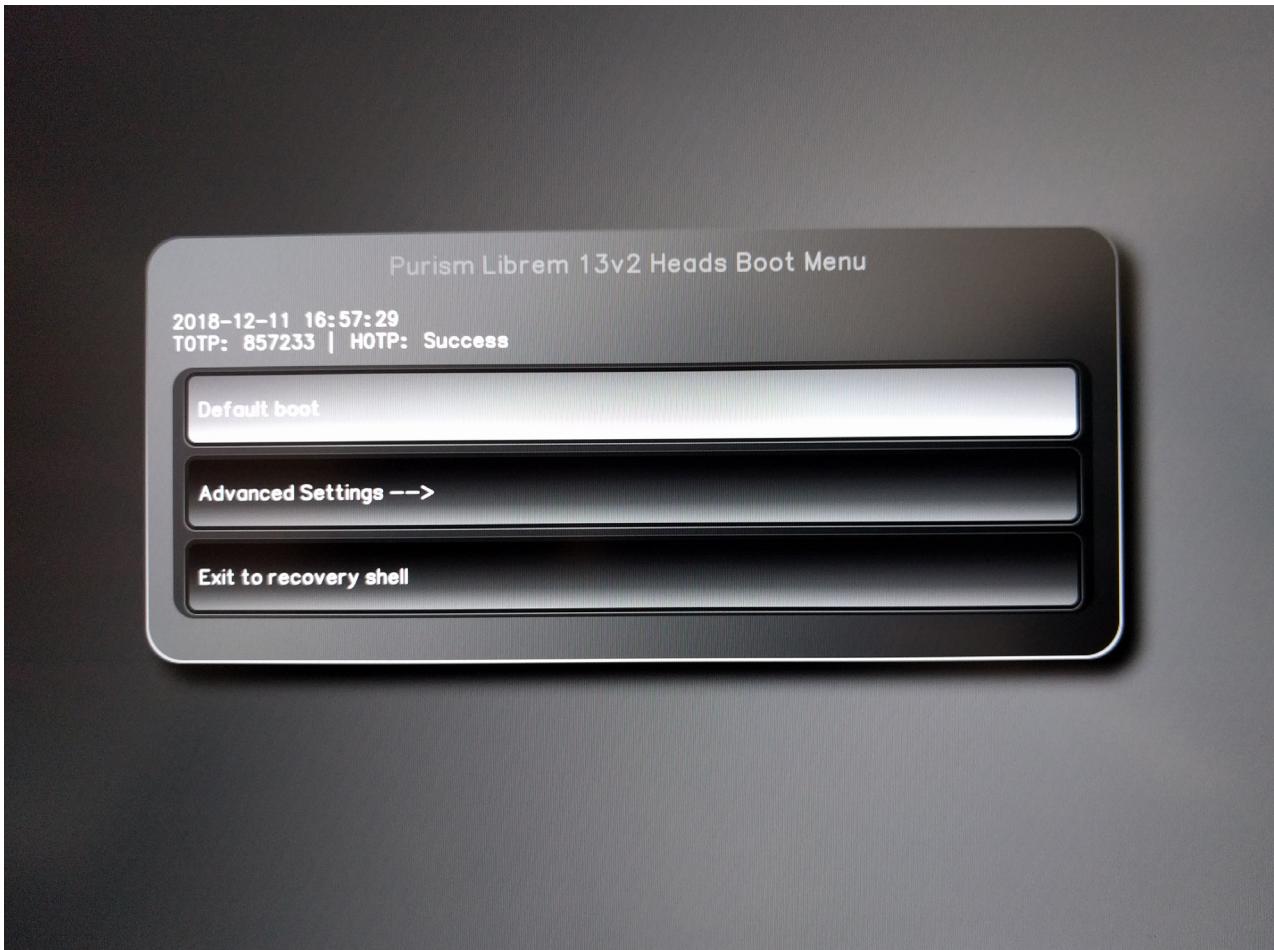


Figure 1. The Default Heads Boot Screen

and Heads would generate a six-digit HOTP code and send it over USB. If it matched the code the Librem Key generated on itself, the Librem Key would blink green; otherwise, it would blink red.

So if an attacker modifies the BIOS, the TPM will generate an error, but what if the attacker then resets the TPM with a different secret using the measurements from the tampered BIOS? Those measurements would match, and the TPM would unlock the new secret, but that secret would generate a different six-digit code from what your phone or Librem Key would generate, and you would know something suspicious was happening. Because the TPM is designed to be a tamper-proof device, you cannot

extract the shared secret from it without providing valid measurements. If you reset the TPM, that secret is also erased.

Boot Security and GPG Keys

Once you have verified that the BIOS is trustworthy, you can move on to booting your OS. But before Heads will boot into the OS, it first checks all of the files within the /boot partition to make sure they haven't changed from when you last signed all of them. When you first set up Heads, you add one or more public GPG keys to a keyring within the Heads runtime environment. Heads provides a mechanism not only to add GPG keys to a standalone Heads coreboot ROM file, but you also can add them to the running BIOS. In that case, Heads actually will pull down a copy of the running BIOS, modify it on the fly, and then reflash it.

Once you have a set of trusted GPG keys in the Heads keyring, you then can sign the files within /boot with your corresponding GPG private key using the Heads GUI. Heads will create a file containing sha256sums for all of the files within /boot and then sign that file with your GPG private key and store the signature in /boot as well. This will require that you have some kind of USB security token that has OpenPGP smartcard support, and Heads will prompt you to insert your USB GPG key whenever you sign these files.

When you tell Heads to boot into your OS, it first gets flashed into the BIOS, and it can read your GRUB config file and provide you with a boot menu based on the options in that config file.

Also, whenever you update or add a new kernel, change an existing initrd file, or modify your GRUB config, Heads will detect the change, showing you an error at the next boot. Along with that error will be an option to re-sign all of the files in /boot, in the case that you changed the files yourself. If you didn't expect those files to change, of course, then this could be a sign of tampering!

Building and Installing Heads

Heads is reproducibly built, which means that it's designed so that if multiple people

were to build the same specific release of Heads with the same build options at different times on different systems, they should get the exact same binary. Because Heads runs on specific BIOS chips, it needs to cross-compile the kernel and other software for that platform, which means that in addition to building a complete Linux kernel and coreboot, you also will need to build a cross-compiler and supporting tools when you build Heads.

Your local system also will need certain system libraries so you can build coreboot and Heads. On a Debian-based system, you can use apt to install them:

```
sudo apt install git build-essential bison flex m4 zlib1g-dev
  gnat libpci-dev libusb-dev libusb-1.0-0-dev dmidecode
  bsdiff python2.7 pv libelf-dev pkg-config cmake
```

For other systems, use your packaging tool to install the equivalent packages for your platform. Once those are installed, the next step is to get the most recent Heads source code and go into the root of that build directory:

```
git clone https://github.com/osresearch/heads.git
cd heads/
```

The next step is to pull down any binary blobs your board might need for coreboot to boot. Go to the blobs/ directory inside Heads, and see if your board has a directory represented in there. If so, `cd` to it and read the instructions for how to pull down your binary blobs for coreboot. For instance, on Librem hardware:

```
cd blobs/librem_skl/
./get_blobs.sh
```

Once you have gotten any blobs you may need, move back to the root of the Heads build directory. From there, you will see a boards/ directory, and within it are directories for each of the motherboards that Heads supports. Each of those boards has a corresponding configuration file inside its respective directory that set

important options, such as what partition to use for /boot and for USB boot devices, what kernel options (if any) to pass along to the OS when it boots, and which init script to load into. These configuration files are already set up for the most part to work with the corresponding motherboard, but you should review the configuration file for your board and confirm in particular that the **CONFIG_BOOT_DEV** and **CONFIG_USB_BOOT_DEV** variables are pointing to the correct /boot and USB boot device, respectively.

Once you are finished editing the configuration file, it's time to build Heads. Change back to the root of the Heads source code and set the particular board with an environment variable while running the **make** command. So for instance, to build for a ThinkPad X230, you would type:

```
make BOARD=x230
```

The first time you build Heads, it will take quite a long time! Just be patient as it builds GCC, coreboot, the Linux kernel and a number of other pieces of software. Subsequent builds will be a lot faster. If the build fails at some point in the process, make a note of what package it was attempting to build, and then check the corresponding build log for that software inside the logs/ directory. More often than not, if you see a build failure for a particular piece of software, it's because you are missing a development library on your system. Reviewing the log file should tell you which libraries are missing.

Once Heads completes the build process, it will dump the corresponding coreboot ROM image into boards/<boardname>/coreboot.rom, so in the case of the above X230 example, it would be in boards/x230/coreboot.rom. You now are ready to install Heads as your BIOS by flashing that ROM image.

Flashing Heads

Once you have built your Heads coreboot ROM, the next step is to flash it over the top of your existing BIOS. How you flash Heads on your computer will vary depending on the specific motherboard you have for a number of reasons. First, each laptop

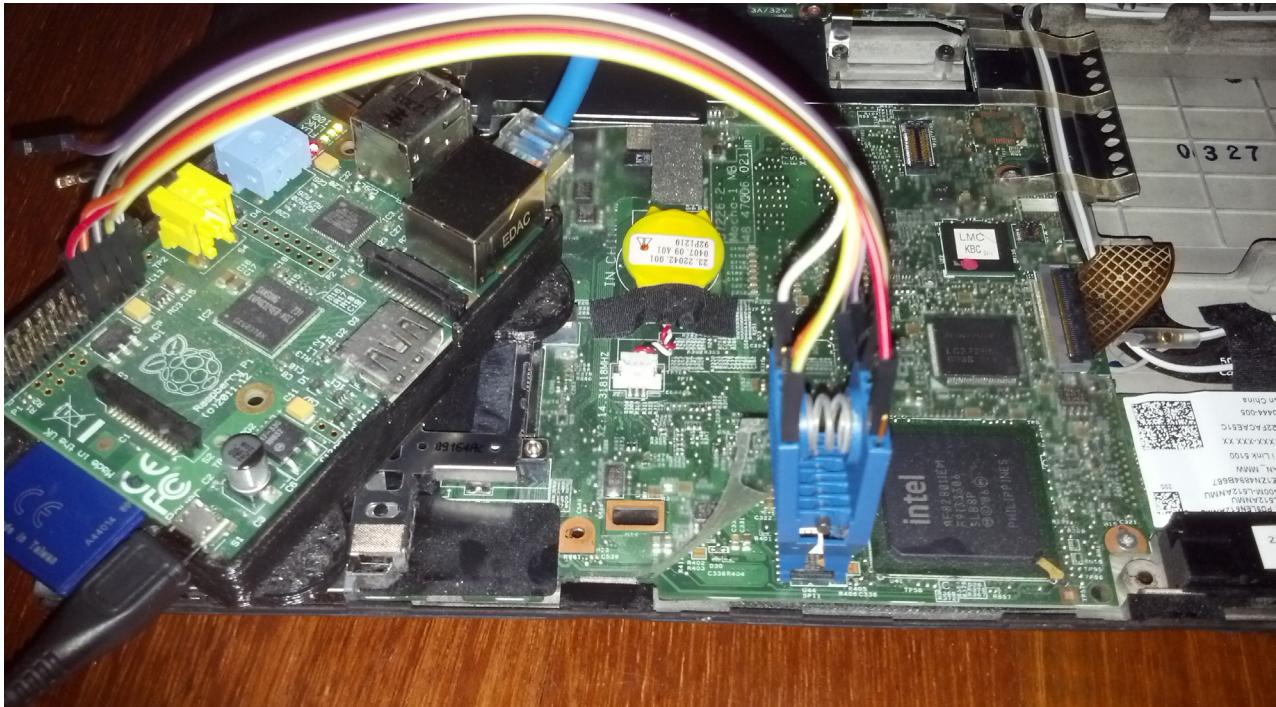


Figure 2. Hardware Flashing a BIOS with a Raspberry Pi

uses its own set of flashrom options that are specific to the BIOS chip it has on board, so you will need to reference the flashrom options appropriate for your board. Check out the initrd/bin/flash.sh script from within the Heads code base for an example script that provides flashrom options for the supported boards. Note that this script is designed to be run from within the Heads environment itself with a relatively new version of flashrom (1.0 or above). Older flash chips (like on the ThinkPad boards) should work with older flashrom versions that you should be able to install via a package on your current Linux distribution, but newer boards (like on the Purism Librem laptops) will require a newer (1.0) flashrom program. In the latter case, Purism provides instructions [here](#) to pull down and build a current flashrom.

Another reason that flashing Heads varies for different platforms is that although you can update coreboot from within your own operating system using flashrom if it is already installed, if coreboot isn't already installed, some laptops require an initial hardware flash. For instance, unless you bought it from a special vendor, the Lenovo

ThinkPad laptops come with a proprietary vendor-provided BIOS instead of coreboot, so they require an initial hardware flash to overwrite the vendor BIOS. This hardware flash means opening the laptop to expose the BIOS chip, connecting a Pomona clip to it that's attached to one of the many hardware platforms that support flashrom, such as a Raspberry Pi or Beaglebone Black. I cover these steps, including how to back up the existing BIOS, in a past Hack and / article: “[Flash ROMs with a Raspberry Pi](#)”.

If your hardware already has coreboot installed, you should be able to install Heads purely from software by running flashrom from within the native OS. For instance, the Purism Librem laptops come with coreboot already installed (and plan to offer Heads as a pre-installed option in the near future), so you can use flashrom from within the regular operating system to flash the Heads BIOS without opening the machine. In this case, you will want to run flashrom first with the **-r** option, so it will pull down a backup of your existing BIOS to store on a USB thumb drive in case you ever want to revert back.

Using Heads

Once you have flashed Heads for the first time and rebooted, Heads will guide you through the initial setup. First you'll be prompted to add at least one public GPG key to the Heads keyring, which will require that you have the public key on some sort of USB thumb drive ending in .asc. Heads will mount the USB drive and find all possible .asc files on the device and then prompt you as to which of them you want to add. Once you have added the key, Heads will reflash the BIOS and reboot.

Once Heads reboots with GPG keys in place, it will get a TPM error, because the TPM has not yet been set up, so it will guide you through setting up a password for your TPM and creating the initial TOTP/HOTP secret. After it reboots another time, you finally should see the default Heads boot menu that lets you select between your default boot option (not yet configured) or opening an Advanced menu of options.

If you select default boot with no default boot option set, it will detect that state and guide you through selecting a boot option. At that point, it also should detect that you have not yet signed any files in /boot, and it also will guide you through that

process (you will need your USB security token containing your GPG private keys at that point).

Once all of the files have been signed and your default boot option has been set, you should be able to treat Heads much like a regular GRUB menu—boot the computer, confirm there are no alerts and just press Enter to boot into your default OS. Note that as you update software on your underlying OS, if your package updates change or add any files to the /boot directory, you'll get an alert the next time you reboot that files may have been tampered with. If you know that this was caused by your package update and not something malicious, you can just re-sign all of the files in /boot with your private GPG key.

You can apply updates to Heads completely within the Heads menu. Within the Advanced options menu is a submenu that allows you to flash the BIOS. Within this menu, you can insert a USB drive containing *.rom files and have Heads flash them over the top of your current Heads ROM. There are two main flashing options: flash a ROM and flash a *cleaned* ROM. The first option is pretty self-explanatory, but in the case of a cleaned ROM, Heads will flash the BIOS, but it won't copy over any existing GPG public keys or other custom changes you may have made to Heads on top of the default ROM. Use this option if you ever want to revert back to a pure factory state (or flash some other non-Heads BIOS), and otherwise use the default flashing option to copy your keyring to the updated Heads ROM.

Conclusion

Although installing and using Heads is not for the faint of heart, if you have experimented with coreboot on systems in the past, it's not that much more complicated. If you want the best in boot security, the effort is definitely worth it, as you will end up with a system that can alert you both to BIOS and kernel-level tampering but with keys completely under your control. ■

Resources

- [The Heads Project](#)
- [UEFI](#)
- [UEFI Secure Boot](#)
- [Intel Trusted Boot](#)
- [TPM \(Trusted Platform Module\)](#)
- [HSM \(Hardware Security Module\)](#)
- [coreboot](#)
- [LinuxBoot](#)
- “FOSS Project Spotlight: LinuxBoot” by David Hendricks, Andrea Barberio and Ron Minnich
- [Lenovo](#)
- [Purism](#)
- “Flash ROMs with a Raspberry Pi” by Kyle Rankin



Kyle Rankin is a Tech Editor and columnist at *Linux Journal* and the Chief Security Officer at Purism. He is the author of *Linux Hardening in Hostile Networks*, *DevOps Troubleshooting*, *The Official Ubuntu Server Book*, *Knoppix Hacks*, *Knoppix Pocket Reference*, *Linux Multimedia Hacks* and *Ubuntu Hacks*, and also a contributor to a number of other O'Reilly books. Rankin speaks frequently on security and open-source software including at BsidesLV, O'Reilly Security Conference, OSCON, SCALE, CactusCon, Linux World Expo and PenguinCon. You can follow him at @kylerankin.

Send comments or feedback
via <http://www.linuxjournal.com/contact>
or email ljeditor@linuxjournal.com.

Programming Text Windows with ncurses

How to use ncurses to manipulate your terminal screen.

By Jim Hall

In my article series about programming for the text console using the ncurses library, I showed you how to draw text on the screen and use basic text attributes. My examples of Sierpinski’s Triangle (see “[Getting Started with ncurses](#)”) and a simple Quest adventure game (see “[Creating an Adventure Game in the Terminal with ncurses](#)”) used the entire screen at once.

But what if it makes more sense to divide the screen into portions? For example, the adventure game might divide the screen to use part of it for the game map and another portion of the screen for the player’s status. Many programs organize the screen into multiple parts—for instance, the Emacs editor uses an editing pane, a status bar and a command bar. You might need to divide your program’s display areas similarly. There’s an easy way to do that, and that’s with the windows functions in ncurses. This is a standard part of any curses-compatible library.

Simple Senet

You may associate “windows” with a graphical environment, but that is not the case here. In ncurses, “windows” are a means to divide the screen into logical areas. Once you define a window, you don’t need to track its location on the screen; you just draw to your window using a set of ncurses functions.

To demonstrate, let me define a game board in an unexpected way. The ancient Egyptian game [Senet](#) uses a board of 30 squares arranged in three rows and ten columns. Two players move their pieces around the board in a backward “S”

formation, so that the board looks like this:

1	2	3	4	5	6	7	8	9	10
20	19	18	17	16	15	14	13	12	11
21	22	23	24	25	26	27	28	29	30

Without the windows functions, you'd have to keep track of the row and column for each piece and draw them separately. Since the board is arranged in a backward “S” pattern, you'll always need to do weird math to position the row and column correctly every time you update each square on the board. But with the windows functions, ncurses lets you define the squares once, including their position, and later refer to those windows by a logical identifier.

The ncurses function `newwin()` lets you define a text window of certain dimensions at a specific location on the screen:

```
WINDOW *newwin(int nlines, int ncols, int begin_y,  
               int begin_x);
```

The `newwin()` function returns a pointer of type `WINDOW*` that you can store in an array for later reference. To create a Senet board, you can use a global array `BOARD[30]`, and write a function to define the 30 squares of the Senet board using windows:

```
#define SQ_HEIGHT 5  
define SQ_WIDTH 8  
  
WINDOW *BOARD[30];  
  
void create_board(void)  
{  
    int i;
```

```
int starty, startx;

starty = 0;
for (i = 0; i < 10; i++) {
    startx = i * SQ_WIDTH;
    BOARD[i] = newwin(SQ_HEIGHT, SQ_WIDTH, starty,
                      startx);
}

starty = SQ_HEIGHT;
for (i = 10; i < 20; i++) {
    startx = (19 - i) * SQ_WIDTH;
    BOARD[i] = newwin(SQ_HEIGHT, SQ_WIDTH, starty,
                      startx);
}

starty = 2 * SQ_HEIGHT;
for (i = 20; i < 30; i++) {
    startx = (i - 20) * SQ_WIDTH;
    BOARD[i] = newwin(SQ_HEIGHT, SQ_WIDTH, starty,
                      startx);
}

/* put border on each window and refresh */

for (i = 0; i < 30; i++) {
    box(BOARD[sq], '2', '2');
    wrefresh(BOARD[sq]);
}
}
```

The first part of this function uses `newwin()` to define the 30 squares. I divided this into three parts to make it obvious that the second row actually counts backward.

Text windows in ncurses don't create a "frame" to show the window on the screen. If you want to draw a frame, you can do so using one of two functions. Here, after defining the windows, the function then calls the ncurses function `box()` to draw the square on the screen. Normally, the `box()` function takes arguments for the characters to use for the vertical and horizontal borders; if you pass zero as either or both arguments, ncurses uses a default line-drawing character.

Note that instead of the usual `refresh()` function to update the screen, you need to use the window-specific `wrefresh()` function to refresh the window.

In the *Senet* game, several squares carry certain meaning. Under common *Senet* rules, players must stop on square 26 (window array element `BOARD[25]`) before they can move off the board. Square 27 (array element `BOARD[26]`) is a trap, which sends the player back to square 15 (`BOARD[14]`). Squares 28 and 29 (`BOARD[27]` and `BOARD[28]`, respectively) require the player to throw a "3" or "2" exactly to move their piece off the board.

To represent these special squares, I replaced the `box()` and `wrefresh()` functions at the end of `draw_board()` with a call to my own function, `draw_square()`. This function draws a different border for the special squares:

```
/* put border on each window and refresh */

for (i = 0; i < 30; i++) {
    draw_square(i);
}

void draw_square(int sq)
{
    switch (sq) {
        case 14: /* revive square */
            wborder(BOARD[sq], '#', '#', '#', '#', '#', '#',
                    '#', '#');
            break;
        case 26: /* trap */
            wbkgnd(BOARD[sq], COLOR_PAIR(1));
            wattron(BOARD[sq], A_BOLD);
            wborder(BOARD[sq], ' ', ' ', ' ', ' ', ' ', ' ',
                    ' ', ' ');
            wattroff(BOARD[sq], A_BOLD);
            wbkgnd(BOARD[sq], COLOR_PAIR(0));
            break;
        case 27: /* square 28 */
            wbkgnd(BOARD[sq], COLOR_PAIR(1));
            wattron(BOARD[sq], A_BOLD);
            wborder(BOARD[sq], ' ', ' ', ' ', ' ', ' ', ' ',
                    ' ', ' ');
            wattroff(BOARD[sq], A_BOLD);
            wbkgnd(BOARD[sq], COLOR_PAIR(0));
            break;
        case 28: /* square 29 */
            wbkgnd(BOARD[sq], COLOR_PAIR(1));
            wattron(BOARD[sq], A_BOLD);
            wborder(BOARD[sq], ' ', ' ', ' ', ' ', ' ', ' ',
                    ' ', ' ');
            wattroff(BOARD[sq], A_BOLD);
            wbkgnd(BOARD[sq], COLOR_PAIR(0));
            break;
        default:
            box(BOARD[sq], 0, 0);
    }
}
```

```
        ← '#', '#');
break;

case 25:           /* stop square */
box(BOARD[sq], 'X', 'x');
break;

case 26:           /* water square */
box(BOARD[sq], '0', 'o');
break;

case 27:           /* 3-move square */
box(BOARD[sq], '3', '3');
break;

case 28:           /* 2-move square */
box(BOARD[sq], '2', '2');
break;

default:
    box(BOARD[sq], 0, 0);
}

wrefresh(BOARD[sq]);
}
```

The `draw_square()` function shows two ways to draw a frame on a text window. I covered the `box()` function earlier. The other method is with the `wborder()` function, which takes separate arguments for the left, right, top and bottom edges of the window, and the upper-left, upper-right, lower-left and lower-right corners. As with `box()`, if you pass zero as any or all of the arguments, ncurses will use a default line-drawing character.

Look in the sample output to see the difference between calling `box()` and `wborder()` with different arguments. I've intentionally used different methods in my *Senet* program to show a few combinations. For example, the “stop” square uses lowercase “x” for the top and bottom borders, and uppercase “X” for the left and right borders. The “revive” square uses `wborder()` to fill the corners, while the other squares are drawn more simply with `box()`. Note that calling `box()` with character arguments still draws line graphic characters for the corners.

Because this uses the windows functions, the `draw_square()` function doesn't need to know the location of each square. That's all tracked by ncurses as an attribute of the 30 windows that make up the 30 squares on the board. Once the program defines



Figure 1. The *Senet* Board with the First Square Highlighted

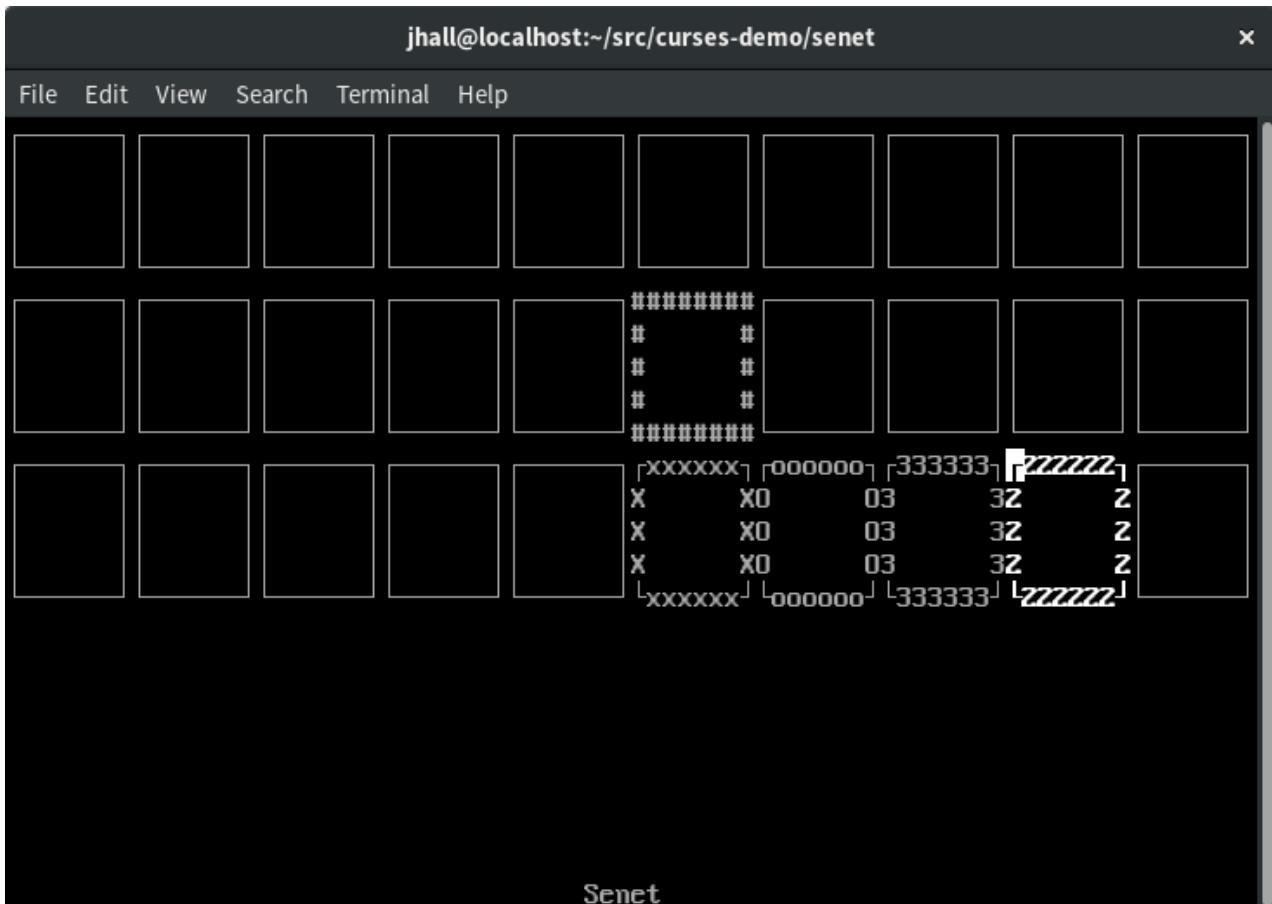


Figure 2. The *Senet* Board with the 29th Square Highlighted

the windows, including their position, drawing to each square is a simple call to an associated “w” function, referencing the window to draw to.

For example, to draw a character in a window, you use the **waddch()** function, or **mvwaddch()**, if you want to draw at a specific location in the window. Most curses functions have a “w” partner function that operates on a specific window, such as these:

```
int move(int y, int x);
int wmove(WINDOW *win, int y, int x);
```

```
int addch(const ctype ch);
int waddch(WINDOW *win, const ctype ch);

int mvaddch(int y, int x, const ctype ch);
int mvwaddch(WINDOW *win, int y, int x, const ctype ch);
```

Full Program

Now that you've seen how to use windows to create 30 independent drawing areas, let's walk through a simple program to draw a *Senet* board and allow the user to navigate through the squares using the plus and minus keys. At a high level, the program follows these steps:

1. Initialize the curses environment.
2. Define and draw the 30 squares on the Senet board.
3. Loop: 1) get a key from the keyboard; 2) adjust the player's location to the previous or next square, accordingly; and 3) repeat.
4. When done, close the curses environment and exit.

Here's the program:

```
/* senet.c */

#include <stdlib.h>
#include <ncurses.h>

#define SQ_HEIGHT 5
#define SQ_WIDTH 8

void create_board(void);
void destroy_board(void);
```

```
void draw_square(int sq);
void highlight_square(int sq);

WINDOW *BOARD[30];

int main(int argc, char **argv)
{
    int key;
    int sq;

    /* initialize curses */

    initscr();
    noecho();
    cbreak();

    if ((LINES < 24) || (COLS < 80)) {
        endwin();
        puts("Your terminal needs to be at least 80x24");
        exit(2);
    }

    /* print welcome text */

    clear();

    mvprintw(LINES - 1, (COLS - 5) / 2, "Senet");
    refresh();

    /* draw board */
```

```
create_board();

/* loop: '+' to increment squares, '-'
   to decrement squares */

sq = 0;
highlight_square(sq);

do {
    key = getch();

    switch (key) {
    case '+':
    case '=':
        if (sq < 29) {
            draw_square(sq);
            highlight_square(++sq);
        }
        break;

    case '-':
    case '_':
        if (sq > 0) {
            draw_square(sq);
            highlight_square(--sq);
        }
    }

} while ((key != 'q') && (key != 'Q'));

/* when done, free up the board, and exit */

destroy_board();
```

```
endwin();
exit(0);
}

void create_board(void)
{
    int i;
    int starty, startx;

    starty = 0;
    for (i = 0; i < 10; i++) {
        startx = i * SQ_WIDTH;
        BOARD[i] = newwin(SQ_HEIGHT, SQ_WIDTH, starty,
                          startx);
    }

    starty = SQ_HEIGHT;
    for (i = 10; i < 20; i++) {
        startx = (19 - i) * SQ_WIDTH;
        BOARD[i] = newwin(SQ_HEIGHT, SQ_WIDTH, starty,
                          startx);
    }

    starty = 2 * SQ_HEIGHT;
    for (i = 20; i < 30; i++) {
        startx = (i - 20) * SQ_WIDTH;
        BOARD[i] = newwin(SQ_HEIGHT, SQ_WIDTH, starty,
                          startx);
    }

/* put border on each window and refresh */
```

```
for (i = 0; i < 30; i++) {
    draw_square(i);
}
}

void destroy_board(void)
{
    int i;

    /* erase every box and delete each window */

    for (i = 0; i < 30; i++) {
        wborder(BOARD[i], ' ', ' ', ' ', ' ', ' ', ' ',
                ' ', ' ');
        wrefresh(BOARD[i]);

        delwin(BOARD[i]);
    }
}

void draw_square(int sq)
{
    switch (sq) {
        case 14:                      /* revive square */
            wborder(BOARD[sq], '#', '#', '#', '#', '#', '#',
                    '#', '#');
            break;

        case 25:                      /* stop square */
            box(BOARD[sq], 'X', 'x');
            break;
    }
}
```

```
case 26: /* water square */
    box(BOARD[sq], '0', 'o');
    break;

case 27: /* 3-move square */
    box(BOARD[sq], '3', '3');
    break;

case 28: /* 2-move square */
    box(BOARD[sq], '2', '2');
    break;

default:
    box(BOARD[sq], 0, 0);
}

wrefresh(BOARD[sq]);
}

void highlight_square(int sq)
{
    wattron(BOARD[sq], A_BOLD);
    draw_square(sq);
    wattroff(BOARD[sq], A_BOLD);
}
```

This is just the bare bones of a *Senet* game. All it does is generate a game board and allow the user to navigate through all of the squares. To keep this focused on the windows functions in ncurses, I've left out all the gameplay and rules.

The program uses only a few functions:

- `void create_board(void);` — defines the 30 squares as text windows and

draws them on the screen.

- `void destroy_board(void);` — erases the 30 squares and deletes the windows.
- `void draw_square(int sq);` — draws a single square on the board. This function draws a different outline depending on the special squares used in Senet.
- `void highlight_square(int sq);` — highlights a square as the user navigates through each square on the board. To keep things simple, this uses the `A_BOLD` attribute instead of color.

Learning on Your Own

This program is a simple example of how to use ncurses windows functions to define separate areas on the screen. The sample program is a game, but you can use this as a starting point for your own programs. Any program that requires updating multiple areas of the screen can use the windows functions.

The ncurses library provides a rich set of functions to update and access the screen in text mode. While graphical user interfaces are very cool, not every program needs to run with a point-and-click interface. If your program runs in plain-text terminals, consider using ncurses to manipulate the terminal screen. ■

Jim Hall is an advocate for free and open-source software, best known for his work on the FreeDOS Project, and he also focuses on the usability of open-source software. Jim is the Chief Information Officer at Ramsey County, Minn.

Resources

- If you are interested in learning more about curses, the ncurses man pages provide extensive documentation on the different functions.
- For more information, including programming examples, read Pradeep Padala's "[NCURSES Programming HOWTO](#)" at the Linux Documentation Project.
- "[Creating an Adventure Game in the Terminal with ncurses](#)" by Jim Hall
- "[Getting Started with ncurses](#)" by Jim Hall
- "[Programming in Color with ncurses](#)" by Jim Hall
- "[About ncurses Colors](#)" by Jim Hall
- [Senet \(Wikipedia\)](#)

Send comments or feedback
via <http://www.linuxjournal.com/contact>
or email ljeditor@linuxjournal.com.

Linux and Open-Source Enthusiasts

SCALE is Back!

Don't miss the next Southern California Linux Expo!
It's North America's largest annual community
organized Open Source gathering.

Register now as a *Linux Journal* subscriber to reserve
your spot and save!

30% OFF Promo Code: LJ35

SOCALLINUXEXPO.ORG

Pasadena Convention Center

MAR 7-10
2019

Open Science, Open Source and R

Free software will save psychology from the Replication Crisis.

By *Andy Wills*

“Study reveals that a lot of psychology research really is just ‘psycho-babble’.”
—*The Independent*.

Psychology changed forever on the August 27, 2015. For the previous four years, the 270 psychologists of the **Open Science Collaboration** had been quietly re-running 100 published psychology experiments. Now, finally, they were ready to share their findings. The results were shocking. Less than half of the re-run experiments had worked.

When someone tries to re-run an experiment, and it doesn't work, we call this a *failure to replicate*. Scientists had known about failures to replicate for a while, but it was only quite recently that the extent of the problem became apparent. Now, an almost existential crisis loomed. That crisis even gained a name: the **Replication Crisis**. Soon, people started asking the same questions about other areas of science. Often, they got similar answers. Only half of results in **economics** replicated. In pre-clinical **cancer studies**, it was worse; only 11% replicated.

Open Science

Clearly, something had to be done. One option would have been to conclude that psychology, economics and parts of medicine could not be studied scientifically. Perhaps those parts of the universe were not lawful in any meaningful way? If so, you shouldn't be surprised if two researchers did the same thing and got different results.

Alternatively, perhaps different researchers got different results because they were doing different things. In most cases, it wasn't possible to tell whether you'd run the experiment exactly the same way as the original authors. This was because all you had to go on was the journal article—a short summary of the methods used and results obtained. If you wanted more detail, you could, in theory, request it from the authors. But, we'd already known for a decade that this approach was seriously broken—in about 70% of cases, **data requests ended in failure**.

Even when the authors sent you their data, it often didn't help that much. One of the most common problems was that when you re-analysed their data, you ended up with different answers to it! This turned out to be quite common, because most descriptions of data analyses provided in journal articles are incomplete and ambiguous. What you really needed was the original authors' source code—an unambiguous and complete record of every data processing step they took, from the raw data files, to the graphs and statistics in the final report. In Psychology in 2015, you almost never could get this.

If you did eventually manage to replicate the authors' analysis, could you be confident that their results were real? Not necessarily. Perhaps they tested only a few people, who were not particularly representative of the population as a whole. In this case, you might want to re-run the experiment yourself, testing a lot more people. Or perhaps the problem was not with their analysis, or their data, but with the method by which they collected their data. For the last 20 years, psychology experiments largely have involved computer-based testing. So, for very many experiments, there is a complete and unambiguous specification of the methods used—the source code for the testing program. But in 2015, this almost never was publicly available either.

In other words, psychology research at the beginning of the Replication Crisis was like closed-source software. You had to take the authors' conclusions entirely on trust, in the same way you have to trust that closed-source software performs as described. There essentially was no way to audit research properly, because you could not access the source code on which the experiment was based—the testing software, the raw data and the analysis scripts.

A growing number of scientists felt this had to change. The year before, in 2014, I had read Stephen Levy's *Hackers*, and from there, I went on to read more about Richard Stallman, Eric S. Raymond and Linus Torvalds. For me, it was a revelation. The Free and Open Source Software community, I felt, showed how science could be different. The pervasiveness of Linux showed that tens of thousands of people with different views and goals could collaborate on a complex project for the common good. Just as important, they could do so without necessarily even having to like each other all the time. That was good news for science. Discussions between academics can get...well, let's just say "heated".

So, in the same way that computing has its advocates of open-source software, psychology and other sciences started gaining advocates for **Open Science**. The phrase Open Science had been coined back in 1998 by Steve Mann, but once the Replication Crisis hit psychology, a lot more of us began to sit up and take notice. Early on, the **Centre for Open Science**, a non-profit company started in 2013, had set up the **Open Science Framework** (OSF). The OSF is a web-based public repository for experiment-related data and code. It's built entirely from free and open-source software.

As awareness of the Replication Crisis grew, peer reviewers started insisting that data and code be made publicly available. Peer review in research is a bit like a code review in IT. Scientists send their articles to a journal for consideration. The journal sends the article out to experts in the field for comment, and the work is accepted for publication only when the journal editor thinks those comments have been adequately addressed. In 2015, Richard Morey and colleagues started the **Peer Reviewers' Openness Initiative**, a declaration that they would not recommend any paper for publication unless it met certain basic standards of open science. Within three years, more than 500 peer reviewers in psychology had signed that declaration.

Open Platforms and R

There's still one major problem to solve. Publishing your scientific source code is essential for open science, but it's not enough. For fully open science, you also need the platforms on which that code runs to be open. Without open platforms, the



future usability of open-source code is at risk. For example, there was a time when many experiments in psychology were written in Microsoft Visual Basic 6 or in Hypercard. Both were closed-source platforms, and neither are now supported by their vendors. It is just not acceptable to have the permanent archival records of science rendered unusable in this way. Equally, it's a pretty narrow

form of Open Science, if only those who can afford to purchase a particular piece of proprietary software are able to access it. All journal articles published in psychology since around 1997 are in PDF format. Academic libraries would not tolerate these archival files being in a proprietary format such as DOCX. We can and must apply the same standards of openness to the platforms on which we base our research.

Psychology has a long history of using closed-source platforms, perhaps most notably the proprietary data analysis software **SPSS**. SPSS initially was released in 1968, and it was acquired by IBM in 2010. Bizarrely, SPSS is such a closed platform, current versions can't even open SPSS output files if they were generated before 2007! Although it's still the most used data analysis software in psychology, its use has been **declining steeply since 2009**. What's been taking up the slack?

In large part, it's **R**. R is a **GNU** project, and so it's free software released under the **GNU General Public Licence**. It works great under Linux, but it also works just fine on Windows and Mac OS too. R is a very long-standing project with great community support. R is also supported by major tech companies, including Microsoft, who maintain the **Microsoft R Application Network**.

R is an implementation of the statistical language S, developed at Bell Labs shortly after UNIX, and inspired by Scheme (a dialect of Lisp). In the 1990s, Ross Ihaka and Robert Gentleman, at the University of Auckland, started to develop R as an open-

source implementation of S. R reached version 1.0 in 2000. In 2004, the R Foundation released R 2.0 and began its annual international conference: `_useR!_`. In 2009, R got its own dedicated journal (*The R Journal*). In 2011, **RStudio** released a desktop and web-based IDE for R. Using R through RStudio is the best option for most new users, although it also works well with Emacs and Vim. The current major release of R landed in 2013, and there are point releases approximately every six months.

I first started using R, on a Mac, in 2012. That was two years before I'd heard of the concept of Free Software, and it was about three years before I ran Linux regularly. So, my choice to move from SPSS to R was not on philosophical grounds. It also was before the Replication Crisis, so I didn't switch for Open Science reasons either. I started using R because it was *just better* than SPSS—a lot better. Scientists spend around **80% of their analysis time on pre-processing**—getting the data into a format where they can apply statistical tests. R is fantastically good at pre-processing, and it's certainly much better than the most common alternative in psychology, which is to pre-process in Microsoft Excel. Data processing in Excel is infamously error-prone. For example, **one in five experiments in genetics have been screwed up by Excel**. Another example: the case for the **UK government's policy of financial austerity** was based on an Excel screw up.

Another great reason for using R is that all analyses take the form of scripts. So, if you have done your analysis completely in R, you already have a full, reproducible record of your analysis path. Anyone with an internet connection can download R and reproduce your analysis using your script. This means we can achieve the goal of fully open, reproducible science really easily with R. This contrasts with the way psychologists mainly use SPSS, which is through a point-and-click interface. It's a fairly common experience that scientists struggle to reproduce their own SPSS-based analysis after a three-month delay. I struggled with this issue myself for years. Although I was always able to reproduce my own analyses eventually, it often took as long to do so as it had the first time around. Since I moved to R, reproducing my own analyses has become as simple as re-running the R script. It also means that now every member of my lab and anyone else I work with can share and audit each other's analyses easily. In many cases, that audit process substantially improves the analysis.

A third thing that makes R so great is that the core language is supplemented by more than 13,000 packages mirrored worldwide on the [Comprehensive R Archive Network](#) (CRAN). Every analysis or graph you can think of is available as a free software package on CRAN. There's even a package to draw graphs in the style of the [xkcd](#) cartoons or in the colour schemes of Wes Anderson movies! In fact, it's so comprehensive, in 2013 the authors of SPSS provided users with the ability to [load R packages within SPSS](#). Their in-house team just couldn't keep up with the breadth and depth of analysis techniques available in R.

R's ability to keep up with the latest techniques in data analysis has been crucial in addressing the Replication Crisis. This is because one of the causes of the Crisis was psychology's reliance on out-dated and easily misinterpreted statistical techniques. Collectively, those techniques are known as Null Hypothesis Significance testing, and they were developed in the early 20th century before the advent of low-cost, high-power computing. Today, we increasingly use more computationally intensive but better techniques, based on Bayes theorem and Monte Carlo techniques. New techniques become available in R years before they're in SPSS. For example, in 2010, Jon Kruschke published a textbook on how to do [Bayesian analysis in R](#). It wasn't until 2017 that [SPSS supported Bayesian analyses](#).

Open Science in R: an Example



form of computer models. For example, one of the classic theories of how we learn to group objects into categories (dogs, cats, bagels and so on) is called ALCOVE. This theory takes the form of a neural network model, which makes predictions about how people will classify objects. We compare those predictions to data from real people making those decisions and evaluate the model on that basis.

In my lab at Plymouth University, we work on the psychology of learning, memory and decision making. In many cases, the theories we are testing are expressed in the

Teaching R

For more than 20 years, teaching statistics in psychology has been synonymous with teaching people how to use SPSS. However, during the last few years, several universities have switched to R, and many more are considering it. One fear about this change was that psychology students would find R harder to learn than SPSS, and that they would like it less. This turns out to be incorrect. In pioneering work by Dale Barr and colleagues at Glasgow University, psychology undergraduates were taught both SPSS and R. They then got to choose which software to use in their final assessments. Around **two-thirds of the students chose R**. Those who chose R also scored higher on the exam. They also scored lower on a standard measure of statistics anxiety. At Plymouth University, new entrants to our Psychology degrees are now just taught R, with SPSS removed from the curriculum entirely. We've seen an increase in what our students can achieve in statistics, while maintaining high levels of student satisfaction.

One of the side benefits of this change, for the R project, is that psychologists tend to be quite good at writing documentation. Andy Field's textbook, *Discovering Statistics*, much-praised by Psychology undergraduates, has had an R version since 2012. More recently, academics have started developing teaching materials that are as open as R is. For example, my own teaching materials, *Research Methods in R*, aimed at first-year psychology undergraduates, are available under a Creative Commons Licence. *Just Enough R*, written by Ben Whalley and aimed at postgraduate students, is available under the same licence.

Traditionally, this computational modelling side to psychology has been fairly closed-source. These models of the mind, which are moderately complex programs, typically are released only as a set of mathematical equations with some explanatory text. The code required to reproduce the results reported is seldom fully published. The result is that it can take several days to several months to reproduce the results of these computer models. The amount of time this wastes is substantial.

Starting in 2016, our lab decided to do something about this issue. Specifically, we released an R package called **catlearn**, short for models of CATegorization and LEARNing. In the current version, released in July 2018, we have implemented nine different models. Like all R packages, the code is open source. The package also includes archives of the full code for simulations of specific experiments and the data sets for those experiments. We're beginning to build a community around the world, with people in the USA, UK, Germany and Switzerland all having contributed code. It's a really exciting time, and I'm looking forward to the release of version 0.7 later this year. If you'd like to contribute, we'd love to hear from you—we desperately need more good programmers. Prior experience of psychology is not essential.

A Final Thought

The Replication Crisis might have been one of the best things ever to happen to psychology. It became a catalyst for much-needed change to our scientific processes. If we can build 21st-century psychology on the principles of Open Science, I think great and enduring discoveries await us. Those future successes will owe a lot to the pioneering example of the Free and Open-Source Software community. Thanks in advance, *Linux Journal* readers! ■

Andy Wills is a Professor of Psychology at the University of Plymouth. He published his [first source code](#), in BBC BASIC, in 1984. Since 1997, he's published around 80 research articles on the psychology and neuroscience of learning and memory. He lives on the south-west coast of England with his wife, daughter and two cats. In his spare time, he plays around with synthesizers. You can find him on Twitter @ajwills72 or via his [website](#).

Resources

- “Study reveals that a lot of psychology research really is just ‘psycho-babble’” by Steve Connor, *The Independent*
- “Estimating the Reproducibility of Psychological Science” by Alexander A. Aarts, Christopher J. Anderson, Joanna E. Anderson and Peter Attridge, *Science*
- Replication Crisis
- “Is Economics Research Replicable? Sixty Published Papers from Thirteen Journals Say ‘Usually Not’” by Andrew C. Chang and Phillip Li, *Finance and Economics Discussion Series 2015-083. Washington: Board of Governors of the Federal Reserve System*
- “Reproducibility: Six red flags for suspect work” by C. Glenn Begley, *Nature*
- “The poor availability of psychological research data for reanalysis” by Jelte Wicherts, Judith Kats, Denny Borsboom and Dylan Molenaar, *American Psychologist*
- *Hackers: Heroes of the Computer Revolution* by Steven Levy, Doubleday, 1984 (Wikipedia)
- Open Science (Wikipedia)
- COS (Center for Open Science)

- [Open Science Framework](#)
- [Peer Reviewers' Openness Initiative](#)
- [SPSS Statistics \(Wikipedia\)](#)
- “[The Popularity of Data Science Software](#)” by Robert A. Muenchen, [r4stats.com](#)
- [The R Project for Statistical Computing](#)
- [The GNU Operating System](#)
- [GNU General Public License](#)
- [Microsoft R Application Network \(MRAN\)](#)
- [RStudio](#)
- “[Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task, Survey Says](#)” by Gil Press, [Forbes](#)
- “[Gene name errors are widespread in the scientific literature](#)” by Mark Ziemann, Yotam Eren and Assam El-Osta, [Genome Biology](#)
- “[Microsoft Excel: The ruiner of global economies?](#)” by Peter Bright, [Ars Technica](#)
- [The Comprehensive R Archive Network](#)

- [xkcd](#)
- “[Calling R from SPSS](#)” by Catherine Dalzell
- [Doing Bayesian Data Analysis](#)
- [Bayesian statistics \(IBM Knowledge Center\)](#)
- [LTC Workshop](#)
- *Discovering Statistics Using R* by Andy Field, Jeremy Miles and Zoe Field, Sage Publishing
- [Research Methods in R](#) by Andy Wills (Teaching Materials)
- [Ben Whalley’s Just Enough R](#)
- [catlearn GitHub Page](#)
- [Acorn Programs](#)

Send comments or feedback
via <https://www.linuxjournal.com/contact>
or email ljeditor@linuxjournal.com.

LINUX JOURNAL

Join the Open-Source Crusade



You subscription includes:

- 12 monthly digital issues
- Fully searchable access to our entire archive (nearly 300 issues)
- Bonus ebook, Sys Admin Fundamentals sent with your paid order

Subscribe.LinuxJournal.com

If Software Is Funded from a Public Source, Its Code Should Be Open Source

If we pay for it, we should be able to use it.

By Glyn Moody



Glyn Moody has been writing about the internet since 1994, and about free software since 1995. In 1997, he wrote the first mainstream feature about GNU/Linux and free software, which appeared in *Wired*. In 2001, his book *Rebel Code: Linux And The Open Source Revolution* was published. Since then, he has written widely about free software and digital rights. He has a [blog](#), and he is active on social media: @glynmoody on [Twitter](#) or [identi.ca](#), and +glynmoody on [Google+](#).

Perhaps because many free software coders have been outsiders and rebels, less attention is paid to the use of open source in government departments than in other contexts. But it's an important battleground, not least because there are special dynamics at play and lots of good reasons to require open-source software. It's unfortunate that the most famous attempt to convert a government IT system from proprietary code to open source—the city of Munich—proved such a difficult experience. Although last year saw [a decision to move back to Windows](#), that seems to be more a failure of IT management, than of the code itself. Moreover, it's worth remembering that the Munich project began back in 2003, when it was a trailblazer. Today, there are [dozens of large-scale migrations](#),

as TechRepublic reports:

Most notable is perhaps the French Gendarmerie, the country's police force, which has switched 70,000 PCs to Gendbuntu, a custom version of the Linux-based OS Ubuntu.

In the same country 15 French ministries have made the switch to using LibreOffice, as has the Dutch Ministry of Defence, while the Italian Ministry of Defence will switch more than 100,000 desktops from Microsoft Office to LibreOffice by 2020 and 25,000 PCs at hospitals in Copenhagen will move from Office to LibreOffice.

More are coming through all the time. The Municipality of Tirana, the biggest in Albania, has just announced it is **moving thousands of desktops to LibreOffice**, and nearly **80% of the city of Barcelona's IT investment this year** will be in open source.

One factor driving this uptake by innovative government departments is the potential to cut costs by avoiding constant upgrade fees. But it's important not to overstate the "free as in beer" element here. All major software projects have associated costs of implementation and support. Departments choosing free software simply because they believe it will save lots of money in obvious ways are likely to be disappointed, and that will be bad for open source's reputation and future projects.

Arguably as important as any cost savings is the use of open standards. This ensures that there is no lock-in to a proprietary solution, and it makes the long-term access and preservation of files much easier. For governments with a broader responsibility to society than simply saving money, that should be a key consideration, even if it hasn't been in the past.

Open-source advocates have rightly noted that free software is a natural fit for any organization that requires solutions based on open standards, interoperability and re-usable components—key elements of the European Commission's new **digital strategy**, for example. One of the leaders here is the UK government. In 2014, it announced a new policy of "**Making things open, making things better**". It achieved this by setting **Open Document Format for Office Applications Version 1.2** as the **default format** for sharing or collaborating with UK government documents. It's

produced an interesting [review of how things have gone in the last four years](#), which concludes:

We cannot have important documents published in formats which do not meet open standards. Government documents are for everyone. Whether you're using Windows, Mac, GNU/Linux, Chrome OS, iOS, Android, or any other system—you have the right to read what we have written and we will continue on our journey to make documents open and accessible.

The use of open standards is not the only big benefit of moving to open source. Another is transparency. Recently it emerged that [Microsoft has been gathering personal information](#) from 300,000 government users of Microsoft Office ProPlus in the Netherlands, without permission and without documentation:

Microsoft systematically collects data on a large scale about the individual use of Word, Excel, PowerPoint and Outlook. Covertly, without informing people. Microsoft does not offer any choice with regard to the amount of data, or possibility to switch off the collection, or ability to see what data are collected, because the data stream is encoded. Similar to this practice in Windows 10, Microsoft has included separate software in the Office software that regularly sends telemetry data to its own servers in the United States.

Moving to open-source solutions does not guarantee that personal data will not leak out, but it does ensure that the problems, once found, can be fixed quickly by government IT departments—something that isn't the case for closed-source products. This is a powerful reason why public funds should mean open source—or as a site created by the Free Software Foundation Europe puts it: [“If it is public money, it should be public code as well”](#).

The site points out some compelling reasons why any government code produced with public money should be free software. They will all be familiar enough to readers of *Linux Journal*. For example, publicly funded code that is released as open source can be used by different departments, and even different governments, to solve

similar problems. That opens the way for feedback and collaboration, producing better code and faster innovation. And open-source code is automatically available to the people who paid for it—members of the public. They too might be able to offer suggestions for improvement, [find bugs](#) or build on it to produce exciting new applications. None of these is possible if government code is kept locked up by companies that write it on behalf of taxpayers.

Once again, the natural fit of open source with public computing is evident. Indeed, when you think about it, it seems ridiculous that public money would be used to produce anything but public code. The Basque Country understood that back in 2012 and brought in a law that required all [software developed for the government there should be released as open source](#). More recently, the Canadian government has made the connection too. Its new [Directive on Management of Information Technology](#) says:

Where possible, use open standards and open source software first.

...

If a custom-built application is the appropriate option, by default any source code written by the government must be released in an open format via Government of Canada websites and services designated by the Treasury Board of Canada Secretariat.

All source code must be released under an appropriate open source software license.

The fact that this approach is not already the norm is something of a failure on the part of the Free Software community. Perhaps it's time to drop the snobbery about open source in government and put more effort into turning it into the next huge win for the world of free software. ■

Send comments or feedback
via <https://www.linuxjournal.com/contact>
or email ljeditor@linuxjournal.com.