Strategies for offline PGP key storage

[LWN subscriber-only content]

October 2, 2017
This article was contributed by Antoine Beaupré

While the adoption of OpenPGP by the general population is marginal at best, it is a critical component for the security community and particularly for Linux distributions. For example, every package uploaded into Debian is verified by the central repository using the maintainer's OpenPGP keys and the repository itself is, in turn, signed using a separate key. If upstream packages also use such signatures, this creates a complete trust path from the original upstream developer to users. Beyond that, pull requests for the Linux kernel are verified using signatures as well. Therefore, the stakes are high: a compromise of the release key, or even of a single maintainer's key, could enable devastating attacks against many machines.

That has led the Debian community to develop a good grasp of best practices for cryptographic signatures (which are typically handled using GNU Privacy Guard, also known as GnuPG or GPG). For example, weak (less than 2048 bits) and vulnerable PGPv3 keys were removed from the keyring in 2015, and there is a strong culture of cross-signing keys between Debian members at in-person meetings. Yet even Debian developers (DDs) do not seem to have established practices on how to actually store critical private key material, as we can see in this discussion on the debian-project mailing list. That email boiled down to a simple request: can I have a "key dongles for dummies" tutorial? Key dongles, or keycards as we'll call them here, are small devices that allow users to store keys on an offline device and provide one possible solution for protecting private key material. In this article, I hope to use my experience in this domain to clarify the issue of how to store those precious private keys that, if compromised, could enable arbitrary code execution on millions of machines all over the world.

Why store keys offline?

Before we go into details about storing keys offline, it may be useful to do a small reminder of how the OpenPGP standard works. OpenPGP keys are made of a main public/private key pair, the certification key, used to sign user identifiers and subkeys. My public key, shown below, has the usual main certification/signature key (marked `SC`) but also an encryption subkey (marked `E`), a separate signature key (`S`), and two authentication keys (marked `A`) which I use as RSA keys to log into servers using SSH, thanks to the Monkeysphere project.

```
pub   rsa4096/792152527B75921E 2009-05-29 [SC] [expires: 2018-04-19]
      8DC901CE64146C048AD50FBB792152527B75921E
uid                 [ultimate] Antoine Beaupré <anarcat@anarc.at>
uid                 [ultimate] Antoine Beaupré <anarcat@koumbit.org>
uid                 [ultimate] Antoine Beaupré <anarcat@orangeseeds.org>
uid                 [ultimate] Antoine Beaupré <anarcat@debian.org>
sub   rsa2048/B7F648FED2DF2587 2012-07-18 [A]
sub   rsa2048/604E4B3EEE02855A 2012-07-20 [A]
sub   rsa4096/A51D5B109C5A5581 2009-05-29 [E]
sub   rsa2048/3EA1DDDDB261D97B 2017-08-23 [S]
```

All the subkeys (`sub`) and identities (`uid`) are bound by the main certification key using cryptographic self-signatures. So while an attacker stealing a private subkey can spoof signatures in my name or authenticate to other servers, that key can always be revoked by the main certification key. But if the certification key gets stolen, all bets are off: the attacker can create or revoke identities or subkeys as they wish. In a catastrophic scenario, an attacker could even steal the key and remove your copies, taking complete control of the key, without any possibility of recovery. Incidentally, this is why it is so important to generate a revocation certificate and store it offline.

So by moving the certification key offline, we reduce the attack surface on the OpenPGP trust chain: day-to-day keys (e.g. email encryption or signature) can stay online but if they get stolen, the certification key can revoke those keys without having to revoke the main certification key as well. Note that a stolen encryption key is a different problem: even if we revoke the encryption subkey, this will only affect future encrypted messages. Previous messages *will* be readable by the attacker with the stolen subkey even if that subkey gets revoked, so the benefits of revoking encryption certificates are more limited.

Common strategies for offline key storage

Considering the security tradeoffs, some propose storing those critical keys offline to reduce those threats. But

where exactly? In an attempt to answer that question, Jonathan McDowell, a member of the Debian keyring maintenance team, said that there are three options: use an external LUKS-encrypted volume, an air-gapped system, or a keycard.

Full-disk encryption like LUKS adds an extra layer of security by hiding the content of the key from an attacker. Even though private keyrings are usually protected by a passphrase, they are easily identifiable as a keyring. But when a volume is fully encrypted, it's not immediately obvious to an attacker there is private key material on the device. According to Sean Whitton, another advantage of LUKS over plain GnuPG keyring encryption is that you can pass the `--iter-time` argument when creating a LUKS partition to increase key-derivation delay, which makes brute-forcing much harder. Indeed, GnuPG 2.x doesn't have a run-time option to configure the key-derivation algorithm, although a patch was introduced recently to make the delay configurable at compile time in `gpg-agent`, which is now responsible for all secret key operations.

The downside of external volumes is complexity: GnuPG makes it difficult to extract secrets out of its keyring, which makes the first setup tricky and error-prone. This is easier in the 2.x series thanks to the new storage system and the associated `keygrip` files, but it still requires arcane knowledge of GPG internals. It is also inconvenient to use secret keys stored outside your main keyring when you actually *do* need to use them, as GPG doesn't know where to find those keys anymore.

Another option is to set up a separate air-gapped system to perform certification operations. An example is the PGP clean room project, which is a live system based on Debian and designed by DD Daniel Pocock to operate an OpenPGP and X.509 certificate authority using commodity hardware. The basic principle is to store the secrets on a different machine that is never connected to the network and, therefore, not exposed to attacks, at least in theory. I have personally discarded that approach because I feel air-gapped systems provide a false sense of security: data eventually does need to come in and out of the system, somehow, even if only to propagate signatures out of the system, which exposes the system to attacks.

System updates are similarly problematic: to keep the system secure, timely security updates need to be deployed to the air-gapped system. A common use pattern is to share data through USB keys, which introduce a vulnerability where attacks like BadUSB can infect the air-gapped system. From there, there is a multitude of exotic ways of exfiltrating the data using LEDs, infrared cameras, or the good old TEMPEST attack. I therefore concluded the complexity tradeoffs of an air-gapped system are not worth it. Furthermore, the workflow for air-gapped systems is complex: even though PGP clean room went a long way, it's still lacking even simple scripts that allow signing or transferring keys, which is a problem shared by the external LUKS storage approach.

Keycards advantages

The approach I have chosen is to use a cryptographic keycard: an external device, usually connected through the USB port, that stores the private key material and performs critical cryptographic operations on the behalf of the host. For example, the FST-01 keycard can perform RSA and ECC public-key decryption without ever exposing the private key material to the host. In effect, a keycard is a miniature computer that performs restricted computations for another host. Keycards usually support multiple "slots" to store subkeys. The OpenPGP standard specifies there are three subkeys available by default: for signature, authentication, and encryption. Finally, keycards can have an actual physical keypad to enter passwords so a potential keylogger cannot capture them, although the keycards I have access to do not feature such a keypad.

We could easily draw a parallel between keycards and an air-gapped system; in effect, a keycard is a miniaturized air-gapped computer and suffers from similar problems. An attacker can intercept data on the host system and attack the device in the same way, if not more easily, because a keycard is actually "online" (i.e. clearly not air-gapped) when connected. The advantage over a fully-fledged air-gapped computer, however, is that the keycard implements only a restricted set of operations. So it is easier to create an open hardware and software design that is audited and verified, which is much harder to accomplish for a general-purpose computer.

Like air-gapped systems, keycards address the scenario where an attacker wants to get the private key material. While an attacker could fool the keycard into signing or decrypting some data, this is possible only while the key is physically connected, and the keycard software will prompt the user for a password before doing the operation, though the keycard can cache the password for some time. In effect, it thwarts offline attacks: to brute-force the key's password, the attacker needs to be on the target system and try to guess the keycard's password, which will lock itself after a limited number of tries. It also provides for a clean and standard interface to store keys offline: a single GnuPG command moves private key material to a keycard (the `keytocard` command in the `--edit-key` interface), whereas moving private key material to a LUKS-encrypted device or air-gapped computer is more complex.

Keycards are also useful if you operate on multiple computers. A common problem when using GnuPG on multiple machines is how to safely copy and synchronize private key material among different devices, which introduces new security problems. Indeed, a "good rule of thumb in a forensics lab", according to Robert J. Hansen on the GnuPG mailing list, is to "store the minimum personal data possible on your systems". Keycards provide the best of both worlds here: you can use your private key on multiple computers without actually storing it in multiple places. In fact, Mike Gerwitz went as far as saying:

> For users that need their GPG key on multiple boxes, I consider a smartcard to be essential. Otherwise, the user is just furthering her risk of compromise.

Keycard tradeoffs

As Gerwitz hinted, there are multiple downsides to using a keycard, however. Another DD, Wouter Verhelst clearly expressed the tradeoffs:

> Smartcards are useful. They ensure that the private half of your key is never on any hard disk or other general storage device, and therefore that it cannot possibly be stolen (because there's only one possible copy of it).
>
> Smartcards are a pain in the ass. They ensure that the private half of your key is never on any hard disk or other general storage device but instead sits in your wallet, so whenever you need to access it, you need to grab your wallet to be able to do so, which takes more effort than just firing up GnuPG. If your laptop doesn't have a builtin cardreader, you also need to fish the reader from your backpack or wherever, etc.

"Smartcards" here refer to older OpenPGP cards that relied on the IEC 7816 smartcard connectors and therefore needed a specially-built smartcard reader. Newer keycards simply use a standard USB connector. In any case, it's true that having an external device introduces new issues: attackers can steal your keycard, you can simply lose it, or wash it with your dirty laundry. A laptop or a computer can also be lost, of course, but it is much easier to lose a small USB keycard than a full laptop — and I have yet to hear of someone shoving a full laptop into a washing machine. When you lose your keycard, unless a separate revocation certificate is available somewhere, you lose complete control of the key, which is catastrophic. But, even if you revoke the lost key, you need to create a new one, which involves rebuilding the web of trust for the key — a rather expensive operation as it usually requires meeting other OpenPGP users in person to exchange fingerprints.

You should therefore think about how to back up the certification key, which is a problem that already exists for online keys; of course, everyone has a revocation certificates and backups of their OpenPGP keys… right? In the keycard scenario, backups may be multiple keycards distributed geographically.

Note that, contrary to an air-gapped system, a key generated on a keycard *cannot* be backed up, by design. For subkeys, this is not a problem as they do not need to be backed up (except encryption keys). But, for a certification key, this means users need to generate the key on the host and transfer it to the keycard, which means the host is expected to have enough entropy to generate cryptographic-strength random numbers, for example. Also consider the possibility of combining different approaches: you could, for example, use a keycard for day-to-day operation, but keep a backup of the certification key on a LUKS-encrypted offline volume.

Keycards introduce a new element into the trust chain: you need to trust the keycard manufacturer to not have any hostile code in the key's firmware or hardware. In addition, you need to trust that the implementation is correct. Keycards are harder to update: the firmware may be deliberately inaccessible to the host for security reasons or may require special software to manipulate. Keycards may be slower than the CPU in performing certain operations because they are small embedded microcontrollers with limited computing power.

Finally, keycards may *encourage* users to trust multiple machines with their secrets, which works against the "minimum personal data" principle. A completely different approach called the trusted physical console (TPC) does the opposite: instead of trying to get private key material onto all of those machines, just have them on a single machine that is used for everything. Unlike a keycard, the TPC is an actual computer, say a laptop, which has the advantage of needing no special procedure to manage keys. The downside is, of course, that you actually need to carry that laptop everywhere you go, which may be problematic, especially in some corporate environments that restrict bringing your own devices.

Quick keycard "howto"

Getting keys onto a keycard is easy enough:

1. Start with a temporary key to test the procedure:

   ```
   export GNUPGHOME=$(mktemp -d)
   gpg --generate-key
   ```

2. Edit the key using its user ID (UID):

   ```
   gpg --edit-key UID
   ```

3. Use the `key` command to select the first subkey, then copy it to the keycard (you can also use the `addcardkey` command to just generate a new subkey directly on the keycard):

   ```
   gpg> key 1
   gpg> keytocard
   ```

4. If you want to *move* the subkey, use the `save` command, which will remove the local copy of the private key, so the keycard will be the *only* copy of the secret key. Otherwise use the `quit` command to save the key on the keycard, but keep the secret key in your normal keyring; answer "n" to "save changes?" and "y" to "quit without saving?" . This way the keycard is a backup of your secret key.

5. Once you are satisfied with the results, repeat steps 1 through 4 with your normal keyring (unset `$GNUPGHOME`)

When a key is moved to a keycard, `--list-secret-keys` will show it as `sec>` (or `ssb>` for subkeys) instead of the usual `sec` keyword. If the key is completely missing (for example, if you moved it to a LUKS container), the `#` sign is used instead. If you need to use a key from a keycard backup, you simply do `gpg --card-edit` with the key plugged in, then type the `fetch` command at the prompt to fetch the public key that corresponds to the private key on the keycard (which stays on the keycard). This is the same procedure as the one to use the secret key on another computer.

Conclusion

There are already informal OpenPGP best-practices guides out there and some recommend storing keys offline, but they rarely explain what exactly that means. Storing your primary secret key offline is important in dealing with possible compromises and we examined the main ways of doing so: either with an air-gapped system, LUKS-encrypted keyring, or by using keycards. Each approach has its own tradeoffs, but I recommend getting familiar with keycards if you use multiple computers and want a standardized interface with minimal configuration trouble.

And of course, those approaches can be combined. This tutorial, for example, uses a keycard on an air-gapped computer, which neatly resolves the question of how to transmit signatures between the air-gapped system and the world. It is definitely not for the faint of heart, however.

Once one has decided to use a keycard, the next order of business is to choose a specific device. That choice will be addressed in a followup article, where I will look at performance, physical design, and other considerations.

Strategies for offline PGP key storage
Posted Oct 3, 2017 9:43 UTC (Tue) by **merge** (subscriber, #65339) [Link]

Why isn't there the concept of a temporary signing key (and certificate), derived from a master key? I'd happily re-upload a new signing key every X months and have it on all my devices when I know it expires. My master identity key could stay super safe and would never have to change. The one extra step of verifying that a current signing (public) key is derived from the one master (public) key doesn't seem too heavy. …but that's easily said without thinking it all through :)

Strategies for offline PGP key storage
Posted Oct 3, 2017 10:32 UTC (Tue) by **grawity** (subscriber, #80596) [Link]

...There is?

You can *already* have a separate subkey for signing files/messages, which expires in a month or two.

The master key is only required for certifying other keys and updates to your own subkeys, e.g. when you need to add a subkey or update the expiry time.

Strategies for offline PGP key storage
Posted Oct 3, 2017 12:10 UTC (Tue) by **ms** (subscriber, #41272) [Link]

Yeah, I have all my subkeys expire every 90 days which lines up with my letsencrypt certs expiring. So once every 80 days I get a reminder to renew everything.

Strategies for offline PGP key storage
Posted Oct 3, 2017 13:28 UTC (Tue) by **merge** (subscriber, #65339) [Link]

That's nice, making use of letsencrypt that way!

So, why use key cards? Creating known-good keys that in turn *can* get compromised until they expire seems more cheap, more safe, and more easy to use, especially when you're at it anyways, regularly finding a place to unlock your most secured files for a very short period of time.

Strategies for offline PGP key storage
Posted Oct 5, 2017 6:48 UTC (Thu) by **madhatter** (subscriber, #4665) [Link]

Because you're probably not the only person that uses your public key. If you're only using gpg to secure your files on your hardware, your point is valid. But if others use your key to correspond with you, and you change it every 80 days, they have a big key validity problem every 80 days.

If instead you have one highly-secure long-lived key that's on a HSM, and you use it to sign your ephemeral encryption keys, then any correspondent who has the public part of your long-lived signing key can get your current public key off any old keyserver and immediately know whether to trust it or not.

Strategies for offline PGP key storage
Posted Oct 5, 2017 7:00 UTC (Thu) by **merge** (subscriber, #65339) [Link]

That's true. But for example Debian encourages to use signing subkeys, see https://wiki.debian.org/Subkeys (although not explicitely short-term keys). But in the end I guess you'd only have to wait until your new signing subkey has landed in all keyrings and let your current one expire, which is solved by overlapping the key validity intervals by a few weeks and always using the oldest.

Strategies for offline PGP key storage
Posted Oct 5, 2017 12:53 UTC (Thu) by **anarcat** (subscriber, #66354) [Link]

One of the problem I've encountered with having multiple signing keys is that not all programs using GPG make it easy to choose which key to use for signing. Last month, for example, I added that signing key and that key took well… about a month to propagate through Debian's infrastructure. That gave me time to notice that:

1. gpg chooses the latest signing subkey (I would have expected it would sign with all available signing subkeys)
2. notmuch-emacs and mutt do not allow you to choose which subkey to use to sign outgoing messages
3. debsign *does* allow you to choose the signing subkey, but that's about the only thing

I had to go back to inline signing to send email… And I had to specify the signing key with a bang ("!") at the end, which was weird and unusual (I would have expected the keygrip to work here for example).

So in short, it's a pain in the back to rotate signing keys, I wouldn't recommend having a workflow based on doing that on a regular basis, unless you control key propagation.

Communicating with an air-gapped system

Posted Oct 3, 2017 9:46 UTC (Tue) by **epa** (subscriber, #39769) [Link]

Wouldn't the old-fashioned serial port be a better choice than USB for getting information to and from your air-gapped system? The serial port can even be constrained in hardware to be output-only, or input-only, just by not connecting some of the pins.

Communicating with an air-gapped system

Posted Oct 3, 2017 11:52 UTC (Tue) by **Funcan** (subscriber, #44209) [Link]

It would, if only I owned a single machine with a serial port…

You can start to look at USB <-> serial converters and such, but really they just become an implementation detail of "design a secure dongle".

Communicating with an air-gapped system

Posted Oct 3, 2017 17:57 UTC (Tue) by **drag** (subscriber, #31333) [Link]

If you want something really dumb and simple and one-way then printing out to QR code and getting brain-dead 2d code scanner may be useful. The simple scanners are essentially just keyboards that type out whatever you scan in + a programmable code (tab key vs return key, etc).

You could print out the master code, destroy the digital copies and just use that. You could even be all cloak and dagger, encrypt the master and split the code up into 2 or more fragments. Keep one half locked in your desk and the second half in a laminated card in your wallet. Or maybe have a 'little black book' of keys you can scan in and then have the password to decrypt them in your wallet.

The downside is that you lose all the features of a proper keycard. The upside is that pretty much everything you need is at your local office supply store.

Communicating with an air-gapped system

Posted Oct 5, 2017 13:17 UTC (Thu) by **genaro** (subscriber, #82632) [Link]

> If you want something really dumb and simple and one-way then printing out to QR code and getting brain-dead 2d code scanner may be useful. The simple scanners are essentially just keyboards that type out whatever you scan in + a programmable code (tab key vs return key, etc).

I did a research paper in college on this topic. It's feasible to export ascii-armored keys and read them with QR. 4096-bit RSA keys are rough, but workable. With newer EC keys the QR method gets much, much easier.

Strategies for offline PGP key storage

Posted Oct 3, 2017 10:11 UTC (Tue) by **ngiger@mus.ch** (subscriber, #4013) [Link]

Did you look at the https://www.crowdsupply.com/nth-dimension/signet. It looks for me like a good compromise between ease to use and privacy.

Strategies for offline PGP key storage

Posted Oct 3, 2017 11:31 UTC (Tue) by **anarcat** (subscriber, #66354) [Link]

I have. Signet is interesting because it runs a similar platform than the FST-01 (STM32L442 microcontroller, while the FST-01 uses STM3F103) so presumably, it may be possible to port Gnuk to it. However, the application deployed on Signet by default is *only* a password manager from what I can tell. Furthermore, Signet is not in production at the time of writing, the crowdfunding is not over yet.

Strategies for offline PGP key storage

Posted Oct 4, 2017 23:35 UTC (Wed) by **nnesse** (guest, #118902) [Link]

Hi, I am the creator of Signet. I just wanted to say that I am very interested in adding more cryptographic functions to the device. It's internal database is flexible enough that PGP key storage could be done as an add-on. There is a bit of a balancing act in terms of the space needed to store GPG keys and algorithms as well as the data and algorithms for password management but I think there is room for it all. This is something I will

probably develop once I've completed all the features I've already promised. I wasn't aware of Gnuk before. I may incorporate it directly or consider making a compatible interface.

Strategies for offline PGP key storage
Posted Oct 3, 2017 13:40 UTC (Tue) by **eahay** (subscriber, #110720) [Link]

Another option is a NitroKey https://www.nitrokey.com

Strategies for offline PGP key storage
Posted Oct 3, 2017 19:54 UTC (Tue) by **dd9jn** (subscriber, #4459) [Link]

Depending on the model Nitrokey either uses the Gnuk software or a standard OpenPGP card from Zeitcontrol for public key operations.

Strategies for offline PGP key storage
Posted Oct 3, 2017 19:58 UTC (Tue) by **anarcat** (subscriber, #66354) [Link]

I encourage people to wait for the next article in the series before discussing the details of all those keycards. In the next article, I will review the Nitrokey PRO, the FST-01, the Yubikey 4 and NEO, including benchmarks and cute graphics. Stay tuned! :)

Strategies for offline PGP key storage
Posted Oct 5, 2017 2:51 UTC (Thu) by **Trelane** (subscriber, #56877) [Link]

awesome! I'm looking forward to it.

Strategies for offline PGP key storage
Posted Oct 5, 2017 6:34 UTC (Thu) by **intrigeri** (subscriber, #82634) [Link]

Excellent, thanks!

Strategies for offline PGP key storage
Posted Oct 3, 2017 22:14 UTC (Tue) by **dsommers** (subscriber, #55274) [Link]

I do have Nitrokey Pro and it worked wonderfully well on my Scientific Linux 7.3 box (not too fast, but I can survive that); it wasn't too easy to get it working, though - I remember I needed some tweaks.

But after I switched to RHEL 7.4, gpg --card-status gives me "Card error" - BUT running openpgp-tool works! So it seems gpg is grumpy about it for some reasons. Anyone got a good idea what could be the issue? I might have forgotten a silly step, but can't figure out what it could be.

Strategies for offline PGP key storage
Posted Oct 5, 2017 4:53 UTC (Thu) by **jans** (guest, #108889) [Link]

I suspect this is related to access restrictions and usually is solved by proper UDEV rules. See these instructions.

Strategies for offline PGP key storage
Posted Oct 5, 2017 9:22 UTC (Thu) by **dsommers** (subscriber, #55274) [Link]

Thank you! I actually had those rules installed. But there was a slight detail I hadn't noticed until your comment. The udev rules uses GROUP="plugdev"; a group name which does not exist on RHEL. Changing that to a group which exists and makes more sense on my setup and it worked.

Again, thank you!

misusing USB keycards?
Posted Oct 4, 2017 16:20 UTC (Wed) by **faramir** (subscriber, #2327) [Link]

If you enable USB on a system so you can use a USB based keycard, aren't you leaving that system open to BadUSB or similar attacks?
If an attacker has control over the computer in which the keycard is installed, they can subvert your data before it is sent to the card. Or simply just use the card directly.

If the keycard caches your password, could they wait until you authenticate to the card and then piggyback on that authentication for their own operations? Is there any indication on the keycard when it is being actively used?

Or maybe they capture the password as you enter it and exfiltrate it. Next time you go to Starbucks, they mug you and steal your keycard as well as your wallet. Depending on how high value a target you are, this seems reasonable. If you are a developer, you might be a much higher value target then you realize; depending on who uses the software that you write.

misusing USB keycards?
Posted Oct 4, 2017 20:52 UTC (Wed) by **anarcat** (subscriber, #66354) [Link]

> If you enable USB on a system so you can use a USB based keycard, aren't you leaving that system

> open to BadUSB or similar ttacks?

Yes, it's one of my core criticism of "airgapped" systems: they are never really airgapped. If you are referring to normal systems, I frankly don't know if you can still run an interactive terminal *without* USB these days. Unless you have a PS/2 mouse and keyboard (and port!), you're pretty much forced to use USB and therefore exposed to that vector anyways.

> If an attacker has control over the computer in which the keycard is installed, they can subvert your data before it is sent to the card. Or simply just use the card directly.

Yep. They can use the card to do any operations it requires. But the point is they can do that only when it's plugged in: the second the key is unplugged, they can't do their evil thing anymore. Furthermore, they can't "steal" the key from you, unless they can find a way to subvert the keycard controller somehow, which is a critical difference with having the key on-disk.

> If the keycard caches your password, could they wait until you authenticate to the card and then piggyback on that authentication for their own operations? Is there any indication on the keycard when it is being actively used?

Yes, they could and no, there's *generally* no visual indicator (although the Yubikey NEO does have a neat little LED in the middle that buzzes when things are happening on the key. It's hardly usable as an indicator, however.

I would rather see a keycard that would force me to tap it to confirm operations. Really, if you're concerned about that level of attacks, you should use one of those card readers that requires a PIN to be entered before operations are allowed on the key.

> Or maybe they capture the password as you enter it and exfiltrate it. Next time you go to Starbucks, they mug you and steal your keycard as well as your wallet. Depending on how high value a target you are, this seems reasonable. If you are a developer, you might be a much higher value target then you realize; depending on who uses the software that you write.

I'm not sure there are such great protections against mugging. Pipewrench cryptography beats any design you can create, really - if that's your threat model, it seems to me you're setting yourself up to failure.

I'm not claiming offline key storage is the silver bullet, but it does solve *some* attack scenarios. The question is if the tradeoffs are worth it for *you*.

misusing USB keycards?
Posted Oct 4, 2017 21:43 UTC (Wed) by **Cyberax** (✭ **supporter** ✭, #52523) [Link]

My Yubikey prompts me to tap on it when it needs to do a U2F or OTP signature.

misusing USB keycards?
Posted Oct 4, 2017 21:44 UTC (Wed) by **karkhaz** (subscriber, #99844) [Link]

> Yes, they could and no, there's *generally* no visual indicator (although the Yubikey NEO does have a neat little LED in the middle that buzzes when things are happening on the key. It's hardly usable as an indicator, however. I would rather see a keycard that would force me to tap it to confirm operations.

Is the touch-to-sign feature on YubiKey 4 what you're looking for?

> YubiKey 4 introduces a new touch feature that allows to protect the use of the private keys with an additional layer. When this functionality is enabled, the result of a cryptographic operation involving a private key (signature, decryption or authentication) is released only if the correct user PIN is provided _and_ the YubiKey touch sensor is triggered

https://developers.yubico.com/PGP/Card_edit.html

misusing USB keycards?
Posted Oct 5, 2017 12:55 UTC (Thu) by **anarcat** (subscriber, #66354) [Link]

That's pretty neat, i gotta say. :)

misusing USB keycards?
Posted Oct 5, 2017 10:44 UTC (Thu) by **tao** (subscriber, #17563) [Link]

I always figured air-gapped meant that the system isn't accessible remotely, not that local attackers aren't able to reach it. If you have local access to hardware, generally all bets are off. An airgapped system isn't connected by WIFI, BT, ethernet, or whatever other means you use to connect to a network, and is preferably kept in a shielded environment. This is the kind of spec needed for things like machines used for signing top level certificates, etc.

The term I'd normally associate with a system that can withstand things like badUSB would be tamper-proof. An ATM, for instance.

Sometimes there's an overlap, and there are degrees of airgapping and tamper-proofing. You probably don't want wifi, BT, etc. for your ATM, but it's definitely connected to the Internet, though hopefully on a VLAN.

misusing USB keycards?

Posted Oct 5, 2017 12:57 UTC (Thu) by **anarcat** (subscriber, #66354) [Link]

if you're connected anyways, where's the gap then?

I could have written a whole article about air-gapped computers - that wasn't my purpose here. It's one of the approaches you can use, and i know it has its merits. the problem is the tradeoffs seem off to me. if you're connected to the internet anyways, how does it differ from a workstation behind a LAN?

the definitions of "air-gapped" sure seem pretty flexible around here… :p which is another problem: if we don't have a clear definition of what an "air gap" is, you're going to have trouble creating a proper threat model analysis…

misusing USB keycards?

Posted Oct 5, 2017 15:05 UTC (Thu) by **nybble41** (subscriber, #55106) [Link]

> if you're connected to the internet anyways, how does it differ from a workstation behind a LAN? … the definitions of "air-gapped" sure seem pretty flexible around here…

It doesn't. You and tao are both saying that an "air-gapped" system is not connected to either the Internet or a LAN. The difference is that tao's definition of "air-gapped" (reasonably, IMHO) does not encompass protection against a local attacker with physical access to the system, e.g. the BadUSB attack. That threat model requires a system which is "tamper-proof", which is a separate consideration from "air-gapped". A "tamper-proof" system can have network links (e.g. ATMs) and an "air-gapped" system can have USB ports. (Suitably restricted, of course—you don't your air-gapped system to automatically establish an Internet connection just because someone plugged a USB network adapter into the port intended for security keys. However, that can be addressed by limiting the USB drivers available, and/or configuring a whitelist of allowed devices.)