Putting pacman in jail: a sandboxing story

Rémi Gacogne, Security Team, Arch Linux

Pass the SALT 2025, July 3rd, 2025

Plan

What is pacman[1] already?

Package manager used by Arch Linux and derivatives.

This is not a talk about the video game, sorry!

Although you can technically use pacman on the Steam Deck!

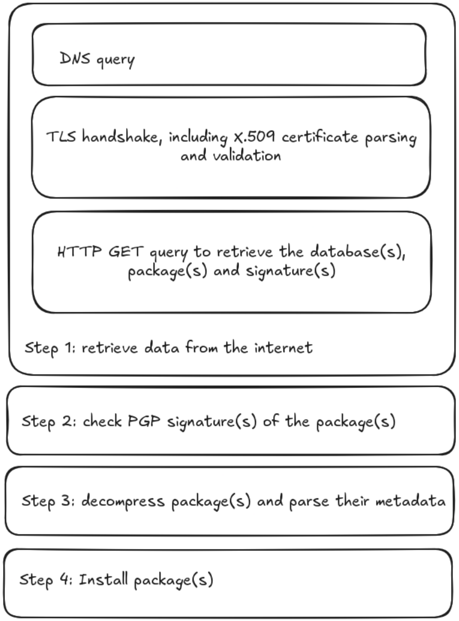
[1] `https://gitlab.archlinux.org/pacman/pacman/`

What does it do?[2]

```
$ sudo pacman -Syu
:: Synchronizing package databases...
 core                     116,3 KiB   969 KiB/s 00:00 [------------] 100%
 extra                      4,7 MiB  4,45 MiB/s 00:00 [-------c o  ]  59%
```

[2] using the “secret” ILoveCandy option

What does it do?

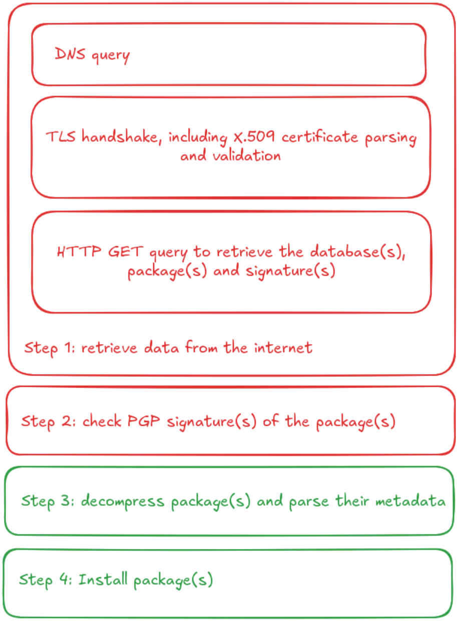

DNS query
TLS handshake, including X.509 certificate parsing and validation
HTTP GET query to retrieve the database(s), package(s) and signature(s)
Step 1: retrieve data from the internet
Step 2: check PGP signature(s) of the package(s)
Step 3: decompress package(s) and parse their metadata
Step 4: Install package(s)

Attack surface

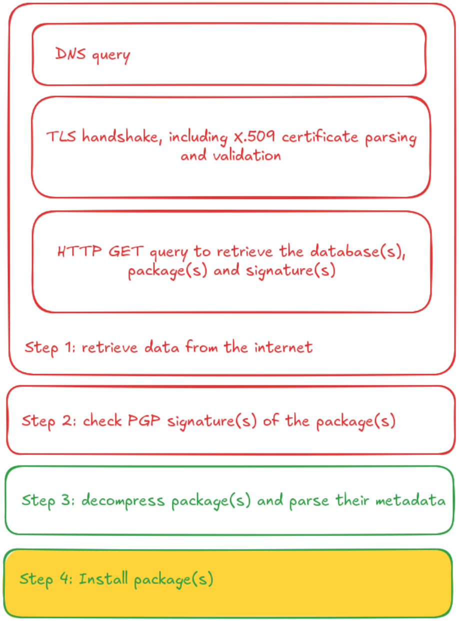
pacman:

- is written in a memory-unsafe language, C
- uses libraries written in memory-unsage languages
- runs as root
- performs complicated tasks involving untrusted content

Attack surface: untrusted content


DNS query
TLS handshake, including X.509 certificate parsing and validation
HTTP GET query to retrieve the database(s), package(s) and signature(s)
Step 1: retrieve data from the internet
Step 2: check PGP signature(s) of the package(s)
Step 3: decompress package(s) and parse their metadata
Step 4: Install package(s)

Privilege separation


DNS query
TLS handshake, including X.509 certificate parsing and validation
HTTP GET query to retrieve the database(s), package(s) and signature(s)
Step 1: retrieve data from the internet
Step 2: check PGP signature(s) of the package(s)
Step 3: decompress package(s) and parse their metadata
Step 4: Install package(s)

Great news!

So, great news, we can sandbox the parts dealing with untrusted content!

- create a new process
- switch to an unprivileged user
- (ideally restrict system calls, filesystem accesses)
- perform the operation dealing with untrusted content
- report back to the initial process
- ?
- profit!

Restricting system calls

Unfortunately Linux does not have an easy way to restrict what a program can do, like OpenBSD's pledge.
What we have is seccomp, which makes it easy to allow/deny specific system call numbers, but involves a fair amount of work to do fine-grained filtering.
Listing allowed system calls is safer, but might break on libc or libcurl upgrades.

Restricting system calls with seccomp

```c
const char *denied_syscalls[] = {
  "kcmp",
  "lookup_dcookie",
  "perf_event_open",
  "pidfd_getfd",
  "ptrace",
  [...]
};
scmp_filter_ctx ctx = seccomp_init(SCMP_ACT_ALLOW);
size_t idx;
for(idx = 0; idx < sizeof(denied_syscalls) / sizeof(*denied_syscalls); idx++) {
  int syscall = seccomp_syscall_resolve_name(denied_syscalls[idx]);
  seccomp_rule_add(ctx, SCMP_ACT_ERRNO(EPERM), syscall, 0);
}
seccomp_load(ctx);
seccomp_release(ctx);
```

Restricting filesystem access

Landlock[3] is a stackable LSM that enables restriction of ambient rights (filesystem or network access).
We can for example use it to mark the whole filesystem as read-only, except for a temporary download directory.

[3] `https://landlock.io/`

Restricting filesystem access with Landlock

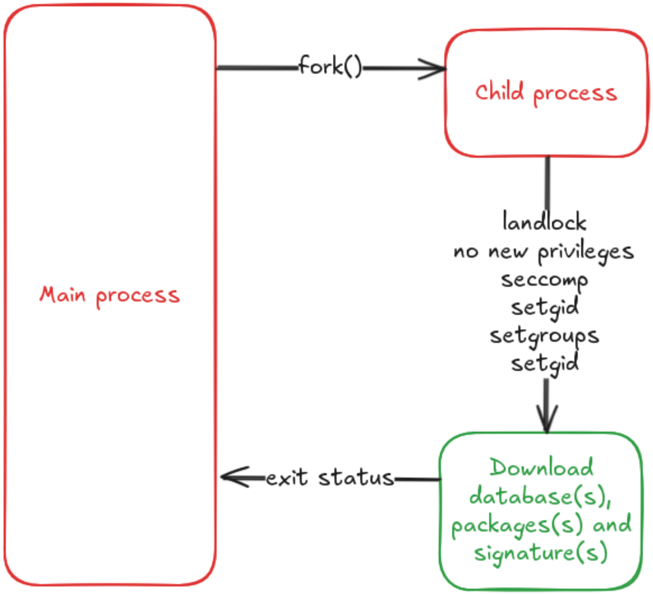
```c
struct landlock_ruleset_attr ruleset_attr = {
  .handled_access_fs = \
  _LANDLOCK_ACCESS_FS_READ | \
  _LANDLOCK_ACCESS_FS_WRITE | \
  _LANDLOCK_ACCESS_FS_REFER | \
  _LANDLOCK_ACCESS_FS_TRUNCATE | \
  LANDLOCK_ACCESS_FS_EXECUTE,
};
struct landlock_path_beneath_attr path_beneath = {
  .allowed_access = _LANDLOCK_ACCESS_FS_READ,
};
int abi = landlock_create_ruleset(NULL, 0, LANDLOCK_CREATE_RULESET_VERSION);
int ruleset_fd = landlock_create_ruleset(&ruleset_attr, sizeof(ruleset_attr), 0);

/* allow / as read-only */
path_beneath.parent_fd = open("/", O_PATH | O_CLOEXEC | O_DIRECTORY);
path_beneath.allowed_access = _LANDLOCK_ACCESS_FS_READ;
landlock_add_rule(ruleset_fd, LANDLOCK_RULE_PATH_BENEATH, &path_beneath, 0);
close(path_beneath.parent_fd);

/* allow read-write access to the directory passed as parameter */
path_beneath.parent_fd = open(path, O_PATH | O_CLOEXEC | O_DIRECTORY);
path_beneath.allowed_access = _LANDLOCK_ACCESS_FS_READ | _LANDLOCK_ACCESS_FS_WRITE |
                              _LANDLOCK_ACCESS_FS_TRUNCATE;

/* make sure allowed_access is a subset of handled_access_fs, which may change for older landlock ABI */
path_beneath.allowed_access &= ruleset_attr.handled_access_fs;

landlock_add_rule(ruleset_fd, LANDLOCK_RULE_PATH_BENEATH, &path_beneath, 0);
landlock_restrict_self(ruleset_fd, 0);

close(path_beneath.parent_fd);
close(ruleset_fd);
```

How hard could it be?

After a few days of coding, I had a working patch!

- create a new process via fork
- switch to an unprivileged user
- restrict system calls via seccomp
- restrict filesystem access via Landlock
- download files to a specific directory
- report back to the initial process via its exit status code

The proof of concept



Submitting the proof of concept

- August 30th, 2021: [pacman-dev] [PATCH] Add optional sandboxing when downloading files
- September 2nd: Positive feedback from Andrew and Allan (but please split this into several patches, remove seccomp and Landlock for now)
- September 5th: second version

Well, crap

Andrew:
"After thinking about this some more, I think this is far too simple. Just running download_internal in an unprivileged fork will break anything that relies on side effects. download_internal sets pm_errno, tracks server errors, and calls a number of front-end callbacks. Losing server error tracking across multiple downloads isn't a big deal, but losing pm_errno is significant and we have no way of knowing what kind of state changes the front-end callbacks might be making. I suspect this would massively break GUI front-ends."

The bad: explanation

It turns out I had missed a very important point: pretty much all my changes are in libalpm, which is a library used not only by pacman, but also by other frontends like paru[4].
They rely on callbacks to interact with libalpm, and some of these are now executed in the unprivileged process, which is not good.

Back to the drawing board.

[4] `https://github.com/morganamilo/paru`

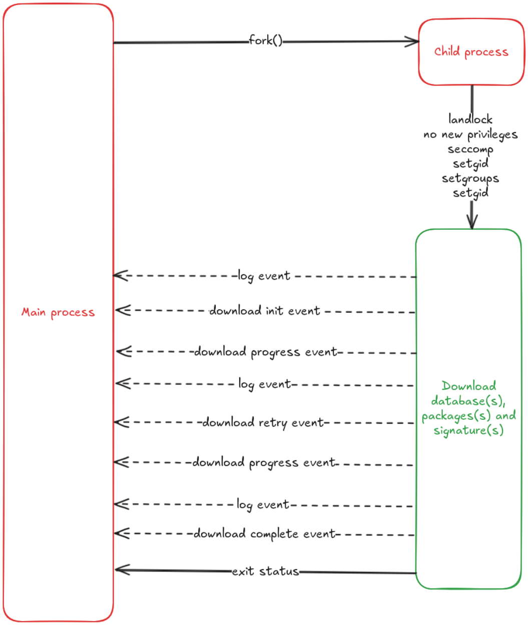
The bad: solution

So we need to pass information back to the parent process: let's set up a pipe.

On October 10th, third version of the patch:

- intercept the callbacks raised in the child process
- serialize them
- send the serialized data over a pipe
- deserialize and process callbacks in the parent process

The bad: solution



Serializing data

- log event: format the string, send the type, level, the length of string then the string itself
- download event: send the type, event, associated struct, filename length then filename

The bad: another privileged parser

The bad news is that we are adding a new parser in a privileged process:

- very simple format
- easy to audit, less than 50 lines of code
- behind a first line of defense

And then we wait

- And then.. nothing happened until August 2022, because both Allan and I were very busy with other projects
- pacman development moved to Arch Linux's gitlab, which made things a lot easier for me
- in November, Allan opened a merge request[5] for the sandboxing
- still a few issues to fix

[5] `https://gitlab.archlinux.org/pacman/pacman/-/merge_requests/23`

Issue 1

Signals were not properly handled in the child, which inherited signal handlers from the parent.

Issue 2

Too much information to pass via the exit status, we settled on passing less information and relying on the presence and state of downloaded files.

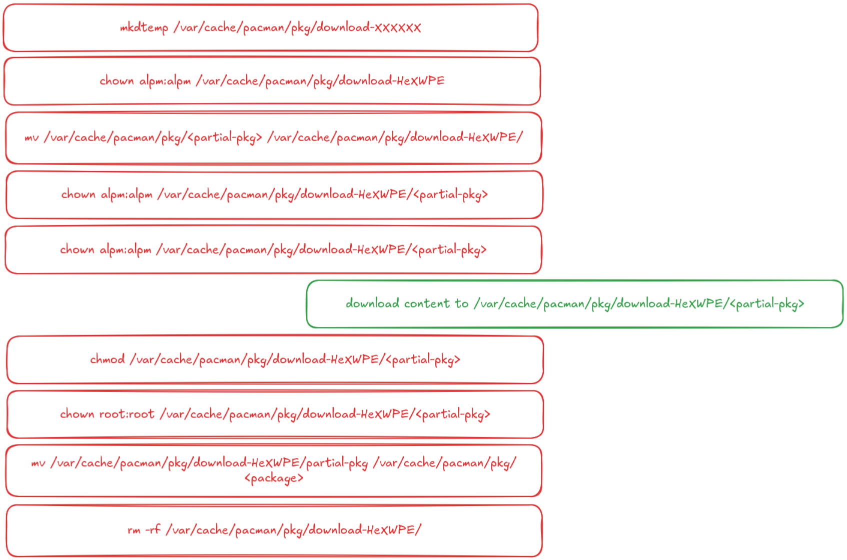
Issue 3

Regression with resuming interrupted downloads, because we could not find them in the next run without knowing their temporary, random names.

Dealing with temporary files

- create a new, temporary download directory via mkdtemp, owned by the unprivileged user
- move any leftover files from a previously interrupted download to that directory
- in the unprivileged child, download files to the temporary directory
- change ownership of successfully downloaded files to root
- move them to the final cache directory
- remove the temporary directory and anything left in it

Dealing with temporary files



```
mkdtemp /var/cache/pacman/pkg/download-XXXXXX
chown alpm:alpm /var/cache/pacman/pkg/download-HeXWPE
mv /var/cache/pacman/pkg/<partial-pkg> /var/cache/pacman/pkg/download-HeXWPE/
chown alpm:alpm /var/cache/pacman/pkg/download-HeXWPE/<partial-pkg>
chown alpm:alpm /var/cache/pacman/pkg/download-HeXWPE/<partial-pkg>
                    download content to /var/cache/pacman/pkg/download-HeXWPE/<partial-pkg>
chmod /var/cache/pacman/pkg/download-HeXWPE/<partial-pkg>
chown root:root /var/cache/pacman/pkg/download-HeXWPE/<partial-pkg>
mv /var/cache/pacman/pkg/download-HeXWPE/partial-pkg /var/cache/pacman/pkg/<package>
rm -rf /var/cache/pacman/pkg/download-HeXWPE/
```

Finally

- April 1st, 2024 (not an April's fool!): it got merged!
- April 5th: Landlock support[6]
- June 17th: Syscall filtering via seccomp [7]
- July 14th: pacman 7.0.0 released "Liechtenstein and Uzbekistan are doubly landlocked"

[6] `https://gitlab.archlinux.org/pacman/pacman/-/merge_requests/167`

[7] `https://gitlab.archlinux.org/pacman/pacman/-/merge_requests/196`

Reminder: before

```
$ sudo pacman -Syu
:: Synchronizing package databases...
 core                  116,3 KiB   969 KiB/s 00:00 [------------] 100%
 extra                   4,7 MiB  4,45 MiB/s 00:00 [-------c o  ]  59%
```

After

```
$ sudo pacman -Syu
:: Synchronizing package databases...
 core                  116,3 KiB   969 KiB/s 00:00 [------------] 100%
 extra                   4,7 MiB  4,45 MiB/s 00:00 [-------c o  ]  59%
```

Mission accomplished!

Profit! .. right?

Downfall

What do you think actually happened?

- regression: unable to download sync database as a user since 7.0[8]
- Sandbox breaks libfakeroot [9]
- Landlock is not supported by the kernel! [10]
- pacman 7.0.0 Landlock issue in systemd-nspawn containers [11]
- Sandboxed dirs seem to interfere with download resumption[12]

[8] `https://gitlab.archlinux.org/pacman/pacman/-/issues/182`
[9] `https://gitlab.archlinux.org/pacman/pacman/-/issues/186`
[10] `https://gitlab.archlinux.org/pacman/pacman/-/issues/190`
[11] `https://gitlab.archlinux.org/pacman/pacman/-/issues/195`
[12] `https://gitlab.archlinux.org/pacman/pacman/-/issues/209`

Summary

- A few unhappy users, but we had workarounds
- Most users are unaware of the change: well done!
- And they are a bit safer today
- Nice side-effect: we managed to get Landlock support enabled in more places!

Next steps

- Sandbox signature verification
- Restrict more syscalls (in particuliar prevent forking while allowing new threads)
- Prevent read access to most of the filesystem

Key takeaway

- sandboxing an existing piece of code is hard, we always underestimate the pain
- you will break some workflows, plan for it: communicate, make it possible to opt-out, be ready to fix what you broke
- if you are writing software in 2025, please design for privilege separation early in the process![13]
- do not mistake the maintainer being busy for lack of interest

[13]yes, some people are working on re-implementing at least parts of libalpm in Rust: `https://github.com/archlinux/alpm`

Thank you! / Questions?