

Moderní izolace procesů v Linuxu: namespaces, seccomp a další

Aneb proč a jak si (ne)vytvořit sandbox

Michal Vyskočil

 vyskocil.me

 [@vyskocilm@twitter.cz](https://twitter.com/vyskocilm)

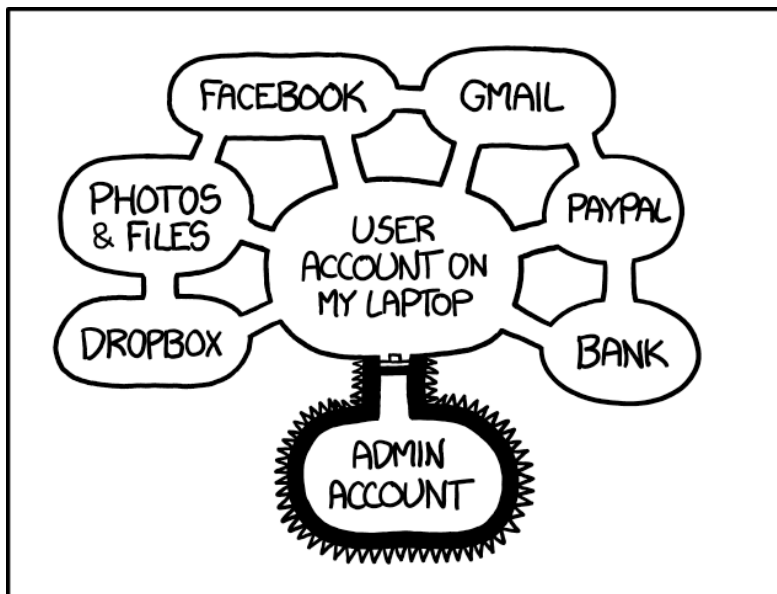
 [LinuxDays 2026](#)

Obsah

1. Je Linux bezpečný?
2. Bezpečnostní technologie Linuxu
3. Ruce na klávesnici
4. Proč je sandbox těžký
5. Můj pokus [snug](#)
6. Q&A

Je Linux bezpečný?

13 let staré xkcd.



IF SOMEONE STEALS MY LAPTOP WHILE I'M
LOGGED IN, THEY CAN READ MY EMAIL, TAKE MY
MONEY, AND IMPERSONATE ME TO MY FRIENDS,
BUT AT LEAST THEY CAN'T INSTALL
DRIVERS WITHOUT MY PERMISSION.

Je vůbec něco bezpečné?

Tohle se stalo v roce 2026 ve čtyřech různých ekosystémech

- **npm, pypi:** Active Exploitation Alert: Shai-Hulud Supply Chain Attack Compromises 100+ Packages with Self-Spreading Malware
- **vs code:** Nx Console VS Code Extension Compromised
- **rust, crates.io:** TrapDoor Proved That Compiling Rust Code Can Be the Attack
- **LLM agenti:** [Data loss] ~221 GB deleted from \$HOME while gpt-5.6-sol sessions were running (codex-cli 0.153.0, macOS 26.6)

Top 5 bezpečnostních problémů

1. Plný přístup

- **Unix:** proces běžící pod uživatelem má plný přístup všude.
- **Problém:** unix aplikace plný přístup *předpokládají*
- **Android model**
 - každá aplikace má vlastní uživatelský účet
 - každá má přístup pouze k vlastním datům a přístup k ostatním je potřeba povolit
 - každá aplikace má vlastní SELinux kontext (Android 5.0+)

Top 5 bezpečnostních problémů

2. Ukládání secrets v čistém textu

- `~/.netrc`
- `~/.aws/credentials`
- `~/.ssh/id_rsa`
- `~/.claude/.credentials.json`
- a spousta dalších

Top 5 bezpečnostních problémů

3. Sdílené proměnné prostředí

- Potomek dědí celé `environ` — `GITHUB_TOKEN`, `AWS_SECRET_ACCESS_KEY`, `SSH_AUTH_SOCK`.
- A `/proc/<pid>/environ` cizího procesu si pod svým uživatelem přečtu taky.

Top 5 bezpečnostních problémů

4. Volný přístup k síti

- Každý proces smí otevřít spojení kamkoli. Ven i dovnitř.
- Na `::` (`127.0.0.` , `0.0.0.0`) visí služby, které autentizaci neřeší, protože jsou *jen lokální*.

Top 5 bezpečnostních problémů

5. Uživatelská (ne)přívětivost bezpečnostních řešení

- Co se nepoužívá snadno, to se nepoužívá!
- **gpg**: konec 90. let, standardně v každé distribuci po tři dekády
- **SELinux**: výchozí ve Fedora od 2004 (22 let), k dispozici v Debian od 2007 (19 let)

Co jde vypnout, se vypne!

- `suid`
- Na druhé straně
- **Let's Encrypt**: HTTP nikdo nepotřebuje od roku 2014
- **ssh**: nahradil rlogin od roku 1995

Bezpečnostní Linuxové Lego

- Distribuce jako Fedora, Ubuntu, openSUSE, Arch Linux, nebo Alpine implementují koncept *unixové pracovní stanice* z konce 80 let.
- Moderní Linux poskytuje mnoho dílků do bezpečnostní skládačky.
- Zároveň musí zůstat kompatibilní s existujícím userspace.

A to je spousta dílků

- **Secure Boot:** Hardware (UEFI) ověřuje kryptografický podpis zavaděče a jádra, čímž brání spuštění podvrženého OS.
- **Measured Boot (TPM 2.0):** Kryptograficky měří a loguje stav každé fáze bootování do čipu TPM pro vzdálenou nebo lokální verifikaci.
- **dm-crypt / LUKS:** Zajišťuje plošné šifrování diskových oddílů na úrovni bloků.
- **dm-verity:** Transparentně ověřuje integritu blokového zařízení
- **IMA (Integrity Measurement Architecture):** Měří a validuje otisky (hash) jednotlivých souborů
- **SELinux / AppArmor:** Mandatory Access Control (MAC) systémy, které globální politikou natvrdo určují, jaké soubory a síťové zdroje smí konkrétní proces navštívit.

A něco i pro běžného uživatele

- **Caps / Capabilities:** Rozbíjejí všemocná práva root na menší, specifická oprávnění (např. `CAP_AUDIT_WRITE` , `CAP_SYS_CHROOT`).
- [CAP_SYS_ADMIN: the new root](#) enters the chat ...
- **Seccomp:** Filtruje systémová volání (syscalls) procesu .
- **Namespaces:** Umožňují mapovat globální prostředky do izolovaného prostředí a tak stavět sandboxy.
- **Landlock:** Moderní, aplikačně orientované LSM, které umožňuje neprivilegovanému procesu se sám omezit.

A co kontejnery?

Kontejnery používají naprosto stejné technologie, jako sandbox.

- User, mount, PID a network namespaces
- `pasta` pro síť u rootless `podman`
- Seccomp — výchozí profil je allowlist v `/usr/share/containers/seccomp.json`
- Capabilities — jsou v tom stejném souboru

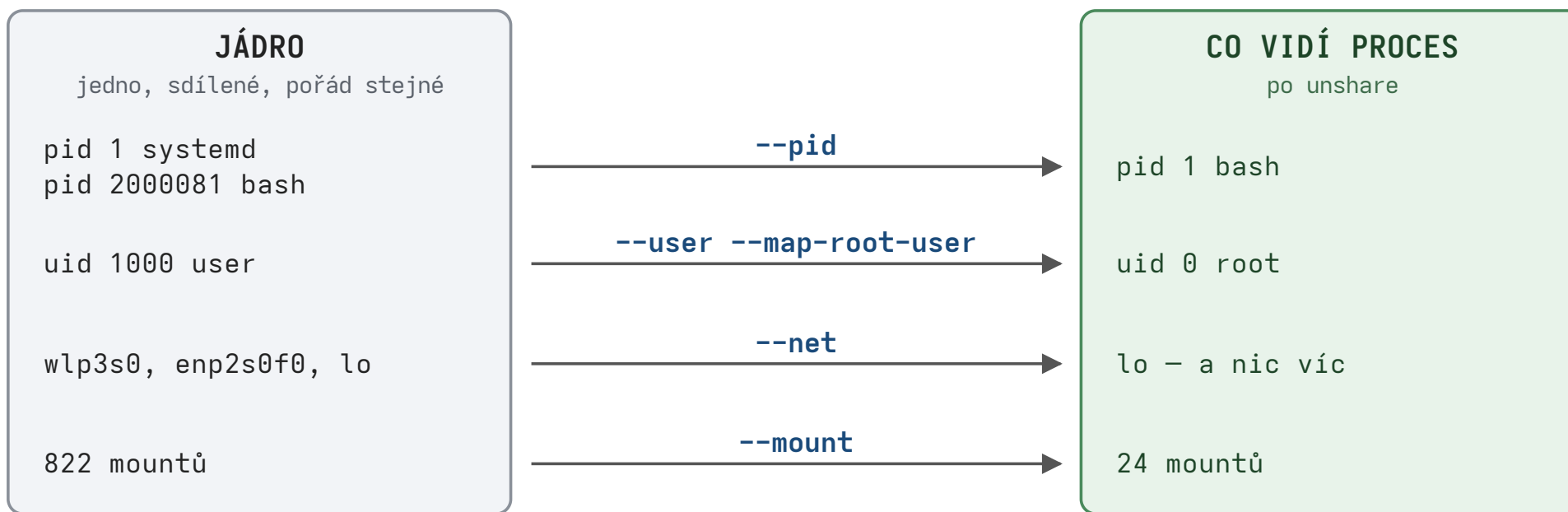
Rozdíl mezi kontejnerem a sandboxem není technologie. Je to *záměr* (a výchozí stav).

Docker

Pokud se mluví o Linux kontejnerech, většina lidí si představí docker. A docker, myšleno dockerd root démon, je *nebezpečný*.

- jeden monolitický démon běžící pod root - nebo ekvivalentním účtem
- zasahuje do nastavení sítě
- a pravidel firewallu
- umožňuje věci jako `--privileged`
- docker-in-docker předává systémového roota *dovnitř* kontejneru

Co je namespace a co dělá unshare



Totéž jádro. Jiný překlad. **Není to virtuálka.**

Ruce na klávesnici: pid namespace

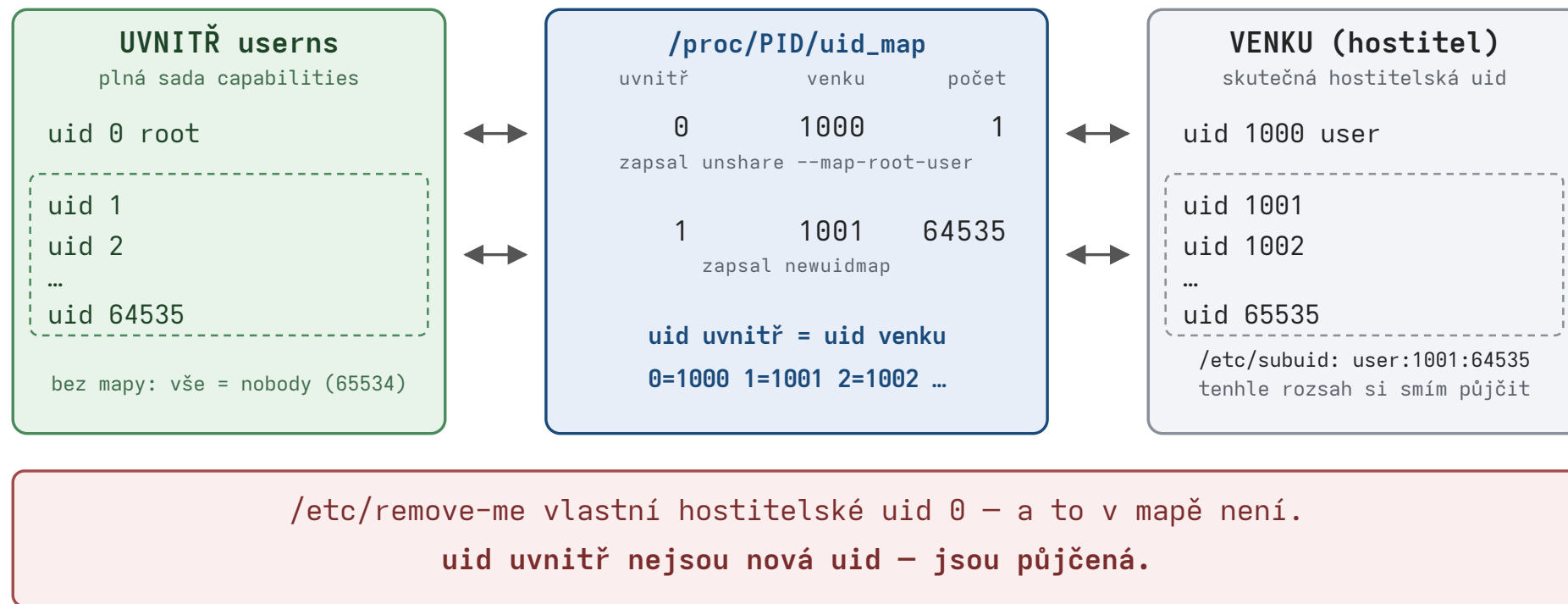
```
# globální stav
$ bash -c 'printf "PID=%d\n" $$'
PID=20000081
# pid namespace spustí bash tak, že si myslím, že je PID=1
$ sudo unshare --pid --fork bash -c 'printf "PID=%d\n" $$'
PID=1
# ale jako běžný uživatel jsme, zdá se, nahraní
$ unshare --pid --fork bash
unshare: unshare failed: Operation not permitted
```

Ruce na klávesnici: user + pid namespace

```
# venku obyčejný uživatel
$ whoami
user
# user namespace smí založit kdokoli – bez sudo, bez hesla
$ unshare --user --pid --fork bash -c 'printf "PID=%d\nUSER=%s\n" $$ $(whoami)'
PID=1
USER=nobody
# ale pořád nedává práva na skutečný systém
$ sudo touch /etc/remove-me
$ unshare --user --fork bash -c 'rm /etc/remove-me'
rm: remove write-protected regular empty file '/etc/remove-me'? y
rm: cannot remove '/etc/remove-me': Permission denied
```

`user namespace` je ten jediný, který nepotřebuje `root`. A přesně na něm stojí všechno ostatní.

user namespace: uvnitř mám jen půjčená uids



/etc/remove-me vlastní hostitelské uid 0 – a to v mapě není.
uid uvnitř nejsou nová uid – jsou půjčená.

Uvnitř mám 64536 uid. Venku je to moje vlastní uid plus rozsah, který mi přidělil správce stroje — a jádro se u každého souboru ptá na to venkovní číslo.

Ruce na klávesnici: unshare a capabilities

```
# capabilities jako uživatel nobody
$ unshare --user grep CapEff /proc/self/status
CapEff: 000000000000000000
# capabilities jako uživatel root
$ unshare --user --map-root-user grep CapEff /proc/self/status
CapEff: 000001ffffffffffff
```

Ruce na klávesnici: jedno uid, nebo celý rozsah

```
# unshare mi namapoval jediné uid – to moje
$ unshare --user --map-root-user cat /proc/self/uid_map
      0      1000      1
$ unshare --user --map-root-user bash -c 'chown 1 f'
chown: changing ownership of 'f': Invalid argument
# podman si zavola newuidmap a ten mu z /etc/subuid přidal celý rozsah
$ podman unshare cat /proc/self/uid_map
      0      1000      1
      1      1001     64535
$ podman unshare bash -c 'chown 1 f; ls -n f'
-rw-r--r--. 1 1 0 0 Sep 21 20:32 f
# a venku ten soubor patří hostitelskému uid 1001
$ ls -n f
-rw-r--r--. 1 1001 1000 0 Sep 21 20:32 f
```

Ruce na klávesnici: unshare a síť

```
# hostitelský systém
$ ip link
...
5: wwan0: <POINTOPOINT,NOARP> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/[519]
# prázdný network namespace má jen loopback
$ unshare --user --net ip link
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
# ale namespaces v Linuxu stále umí překvapit
$ unshare --user --net ls /sys/class/net
enp2s0f0  enp6s0  lo  wlp3s0  wwan0
```

Ruce na klávesnici: mount namespaces a není unshare, jako unshare

```
$ mount | wc -l
820
# unshare --mount nic neodsílí
$ unshare --user --map-root-user --mount \
    bash -c 'mount | wc -l'
820
# protože mount namespace jen zajistí to, že změny uvnitř namespace
# zůstanou viditelné pouze v rámci stejného namespace
$ unshare --user --map-root-user --mount \
    sh -c 'mount -t tmpfs none /home/user && ls -a /home/user/'
. . .
```

Ruce na klávesnici: Seccomp ze shellu

Ukázat seccomp ze shellu je složitější, ale dá se.

```
# bez filtru rm normálně smaže
$ systemd-run --user --pipe \
    sh -c 'touch /tmp/sc && rm /tmp/sc && echo DELETED'
DELETED

# se zákazem unlink/unlinkat – rm ten syscall nedostane
$ systemd-run --user --pipe \
    -p SystemCallFilter='~unlink unlinkat' -p SystemCallErrorNumber=EPERM \
    sh -c 'touch /tmp/sc && rm -f /tmp/sc || echo BLOCKED'
rm: cannot remove '/tmp/sc': Operation not permitted
BLOCKED
```

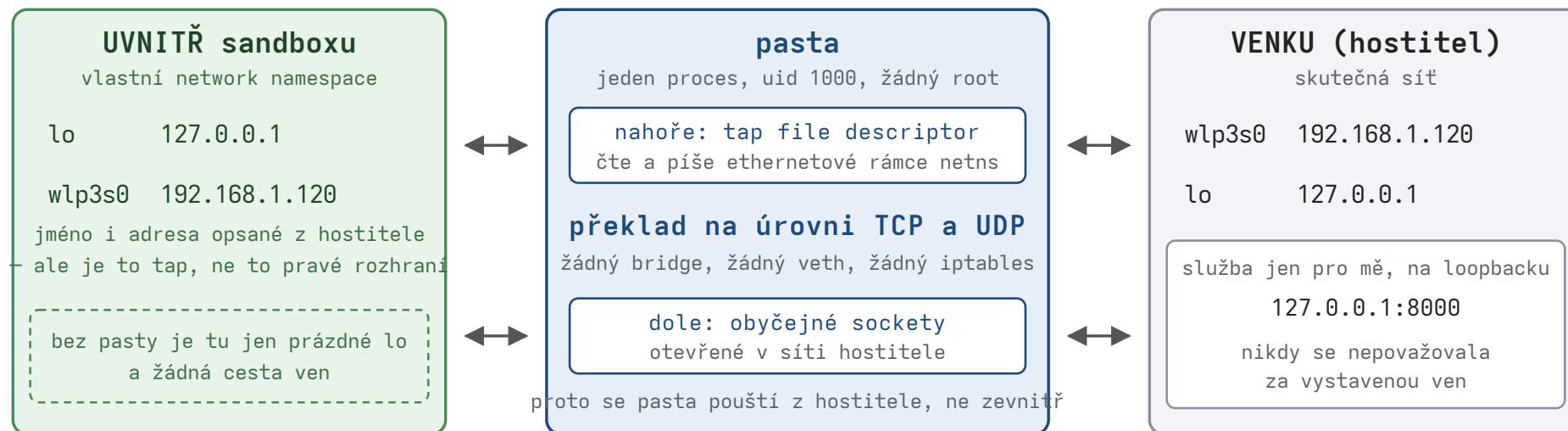
Ruce na klávesnici: user networking **pasta**

```
# na hostu běží "tajná" služba jen na loopbacku
$ python3 -m http.server 8000 --bind 127.0.0.1
# jiný shell
$ curl -s 127.0.0.1:8000/secret
TAJNY-TOKEN-42

# prázdná net namespace na tu službu vůbec nedosáhne
$ unshare --user --net curl 127.0.0.1:8000/secret
curl: (7) Failed to connect to 127.0.0.1:8000 after 0 ms: Could not connect to server

# pasta dá stack – a s ním i host loopback, aniž bych o to řekl
$ pasta --config-net --map-host-loopback none -- curl -s 127.0.0.1:8000/secret
TAJNY-TOKEN-42
```

pasta : síťový stack v userspace



Výchozí `-T auto -U auto`: co poslouchá na hostiteli, je slyšet i uvnitř – i na 127.0.0.1.
pasta chce, aby se kontejner choval jako hostitel. Sandbox chce pravý opak.

pasta není bridge ani NAT. Je to proces, který jednou nohou stojí v prázdné netns a druhou v síti hostitele — a překládá spojení mezi nimi.

Blíže k sandboxu: bubblewrap

- <https://github.com/containers/bubblewrap>
- *Low-level unprivileged sandboxing tool used by Flatpak*
- A narozdíl od `unshare` a `pasta` jej lze použít na vytvoření jednoduchého sandboxu.
- Používá stejné stavební bloky, které jsme si už představili

Ruce na klávesnici: bwrap

Sestavit primitivní sandbox je jednoduchost sama, stačí totiž

1. `unshare(CLONE_NEWNS|CLONE_NEWUSER)` vlastní mount namespace
2. `mount -t tmpfs` prázdný nový kořen
3. `--ro-bind, --bind, --symlink, --proc` navěsit, co má být vidět
4. `pivot_root + umount` starého kořene původní strom přestane existovat

```
$ bwrap --ro-bind /usr /usr --ro-bind /etc /etc \  
        --symlink usr/bin /bin --symlink usr/lib64 /lib64 \  
        --proc /proc --dev /dev --unshare-all \  
        /bin/sh -c 'cut -d" " -f5,9,10 /proc/self/mountinfo | head -1; ls /'  
/ tmpfs rw,seclabel,uid=1000,gid=1000,inode64  
bin dev etc lib64 proc usr
```

Yes, but (1/4): prostředí jde dovnitř celé

```
# $BW je ten bwrap z minulého slidu. Sandbox, co vypadal hotově.  
$ $BW sh -c 'env | wc -l'  
105  
$ $BW sh -c 'env | grep -o "^[A-Z_]*" | grep "SSH\|AUTH\|TOKEN"'  
SSH_AUTH_SOCKET  
XAUTHORITY  
CLAUDE_CODE_MESSAGING_TOKEN
```

- Sandbox, který řeší jen soubory, tohle mine. Token není soubor.

Yes, but (2/4): i otevřené file descriptors

```
# uvnitř ta cesta neexistuje, /home není nabindované
$ $BW sh -c 'cat ~/demo-secret/credentials'
cat: /home/user/demo-secret/credentials: No such file or directory
# ale otevřu ten soubor PŘED spuštěním sandboxu
$ exec 9< ~/demo-secret/credentials
$ $BW sh -c 'cat /proc/self/fd/9'
aws_secret_access_key = TAJNY-KLIC-42
$ $BW sh -c 'readlink /proc/self/fd/9'
/home/user/demo-secret/credentials
```

Yes, but (3/4): síť je všechno, nebo nic

```
# --unshare-all: žádná síť. Bezpečné a nepoužitelné.  
$ $BW sh -c 'curl -s --max-time 2 127.0.0.1:8000/secret || echo "curl selhal"'  
curl selhal  
# přidám pasta, ať sandbox síť má  
$ pasta --config-net -- $BW --share-net sh -c 'curl -s 127.0.0.1:8000/secret'  
TAJNY-TOKEN-42  
# „tak tu hostitelskou loopback vypnu"  
$ pasta --config-net --map-host-loopback none -- $BW --share-net sh -c '...'  
TAJNY-TOKEN-42
```

Yes, but (4/4): potřebuju docker

```
# 1. v sandboxu tak, jak jsme ho postavili
$ $BW sh -c 'podman info'
Failed to obtain podman configuration: lstat /run: no such file or directory
# 2. tak mu /run půjčím. Read-only, ať jsem opatrný.
$ $BW --ro-bind /run /run sh -c 'podman info'
... chmod /run/user/1000/libpod: read-only file system
# 3. tak zapisovatelně. A /home, /var i /sys, ať to má všechno.
$ $BW --bind /run /run --bind /home /home \
    --bind /var /var --ro-bind /sys /sys sh -c 'podman info'
msg="running `/usr/bin/newuidmap 16 0 1000 1 1 1001 64535`:
    newuidmap: write to uid_map failed: Operation not permitted"
Error: ... unable to create a new pause process
```

Yes, but: dokončení

- a ještě: kořen je zapisovatelná `tmpfs` , `Seccomp: 0` , úklid po `SIGKILL`
- `docker run ... cat /secret/dir/on/host` má přístup k hostu
- a co kernel keyring
- potřebuji ssh — ale ne celý ssh-agent
- potřebuji identity — jeden klíč, jeden účet
- sandbox, ze kterého se lze dostat, není sandbox
- a zrovna LLM jsou *skvělé* v tom, jak se z něj dostávat

snug

Ted' následuje reklama

*fitting closely and comfortably
marked by cordiality and secure privacy
offering safe concealment
a small private room in a pub*

[Merriam-Webster on snug](#)

Není to jen teoretický problém

- Přednáška začíná popisem Shai-Hulud Supply Chain Attack
- ○ Google: Jak vyrobit prezentaci z markdown?
- ○ `npm install @marp-team/marp-cli`
- ○ 🙈

Praktická ukázka: prezentace pro Linux Days

```
# ~/.config/snug/profiles.d/marp.toml
```

```
[profile.marp]
```

```
description = "A Marp deck: node from /usr, the deck read-only from the parent, directory dist writable."
```

```
include = ["@sys", "@home", "@target-rw", "@parent-ro"]
```

```
rw = ["{target_parent}/dist"]
```

```
[profile.marp.envIRON.set]
```

```
NPM_CONFIG_CACHE = "{target}/.cache/npm"
```

```
PUPPETEER_CACHE_DIR = "{target}/.cache/puppeteer"
```

snug: všechno zakázáno

```
# žádná síť
$ snug -p marp . -- curl https://example.com
curl: (6) Could not resolve host: example.com
# ani ssh-agent
$ snug -p marp . -- ssh-add -l
Could not open a connection to your authentication agent.
# ani AI
$ snug -p marp . -- claude
bwrap: execvp claude: No such file or directory
```

snug to the rescue

Celý skript spouštějící build je mnohem složitější, ale jádro je víceméně spustit tohle

```
snug -p marp . \  
  -- env CHROME_NO_SANDBOX=1 \  
  "$marp" --no-stdin "../$deck" -o "../dist/$base.pdf"
```

Vlastnosti 1/4

`snug` je *experimentální* program na vytváření sandboxovaných prostředí pro Linux

<https://github.com/gomoni/snug>

- nepotřebuje práva `root` ani démona — jedna binárka. Spustí se i z kontejneru (`distrobox` , `toolbox`) a otestuje v běžném CI
- deklarativní, znovupoužitelná a relativně snadná konfigurace — profil umí sandbox jen rozšířit, nikdy zúžit, a na pořadí nezáleží
- skládá dohromady výstup z `getent` , `bwrap` , `pasta` a `podman` — ty spouští sám. Engine navíc vybere `crun` ; `conmon` , `netavark` , `aardvark-dns` a `catatonit` si `podman` najde sám, `snug` je jen zkontroluje a zpřístupní

Vlastnosti 2/4

- vlastní síť — na hostitele se ze sandboxu nedá připojit na žádné jeho adrese, ani na službu naslouchající na `::`
- po `SIGKILL` nezůstane nic — žádný `snug stop`, žádný stale stav, žádné `pid` soubory
- vysvětluje do nejmenších podrobností, jak přesně se sandbox staví (`--dry-run`, `--explain`, `snug doctor`, `snug fix`)

Vlastnosti 3/4

- podman engine uvnitř sandboxu: vlastní netns, pid, ipc a uts. Mezi klientem a enginem filtrující proxy, `podman build` jen s filtrovanou sadou options
- secrets: žádný credential dovnitř — capability jen pod profilem, který ji jmenuje. `--dry-run` vypíše každé tajemství, které run dovnitř položí, i kam
- ssh klíč dovnitř nikdy nevstoupí — agent proxy odpoví připnutými klíči a podepíše jen pro něj. `git+https` zatím moc podporováno není.

Vlastnosti 4/4

- identity — jeden účet na sandbox: klíč, generovaný `~/.gitconfig` a ssh config
- `@claude` — binárka a skills read-only, credentials jako zapisovatelné kopie, nedostupné z jiné session
- HTTP dveře se deklarují jménem, ne portem. Socket otevře host a payload dostane jen deskriptor — a otevře je člověk přes `snug proxy`, ne kód uvnitř

UX sem: snug doctor

 programs – the binaries this host has to provide

- ✓ bubblewrap 0.11.2
 - /usr/bin/bwrap
- ✓ pasta 20260612.a9c61ff-1.3
 - /usr/bin/pasta
- ✓ podman client is usable inside a sandbox
 - /usr/bin/podman
- ✓ podman's helper binaries are all findable
 - common /usr/bin/common
 - netavark /usr/libexec/podman/netavark
 - aardvark-dns /usr/libexec/podman/aardvark-dns
 - catatonit /usr/libexec/podman/catatonit
 - crun /usr/bin/crun

 kernel – what this kernel lets snug build

- ✓ the 5 kernel knobs snug's threat model inherits from this host are set
- ✓ unprivileged user namespaces work
- ✓ an unshared network namespace comes up empty – loopback and nothing else
 -  what a RUN can reach is the profiles' answer, not this probe's – `snug config`
- ✓ snug's own stage starts – clone, uid map, loopback, netns move, re-exec
- ✓ TIOCSSTI disabled kernel-wide – job control works inside the sandbox

 host configuration – files and settings outside the kernel

- ✓ a delegated subuid/subgid range the container engine can use
-  running inside a container (distrobox/podman) – supported
- ✓ profiles load cleanly

 This host can run snug.

UX tam: snug fix

```
snug fix subuid user
user:499:65037
snug: nothing was changed – run `snug fix subuid lp -w` (as root) to append this to /etc/subuid and /etc/subgid
snug: not the conventional 100000 – this namespace's uid_map cannot map it
```

A zase UX

```
$ snug -p marp --explain
snug - what this sandbox would be. Nothing was started.
```

The project directory, writable from inside:

```
/home/user/somedir/linuxdays-2026/marp
```

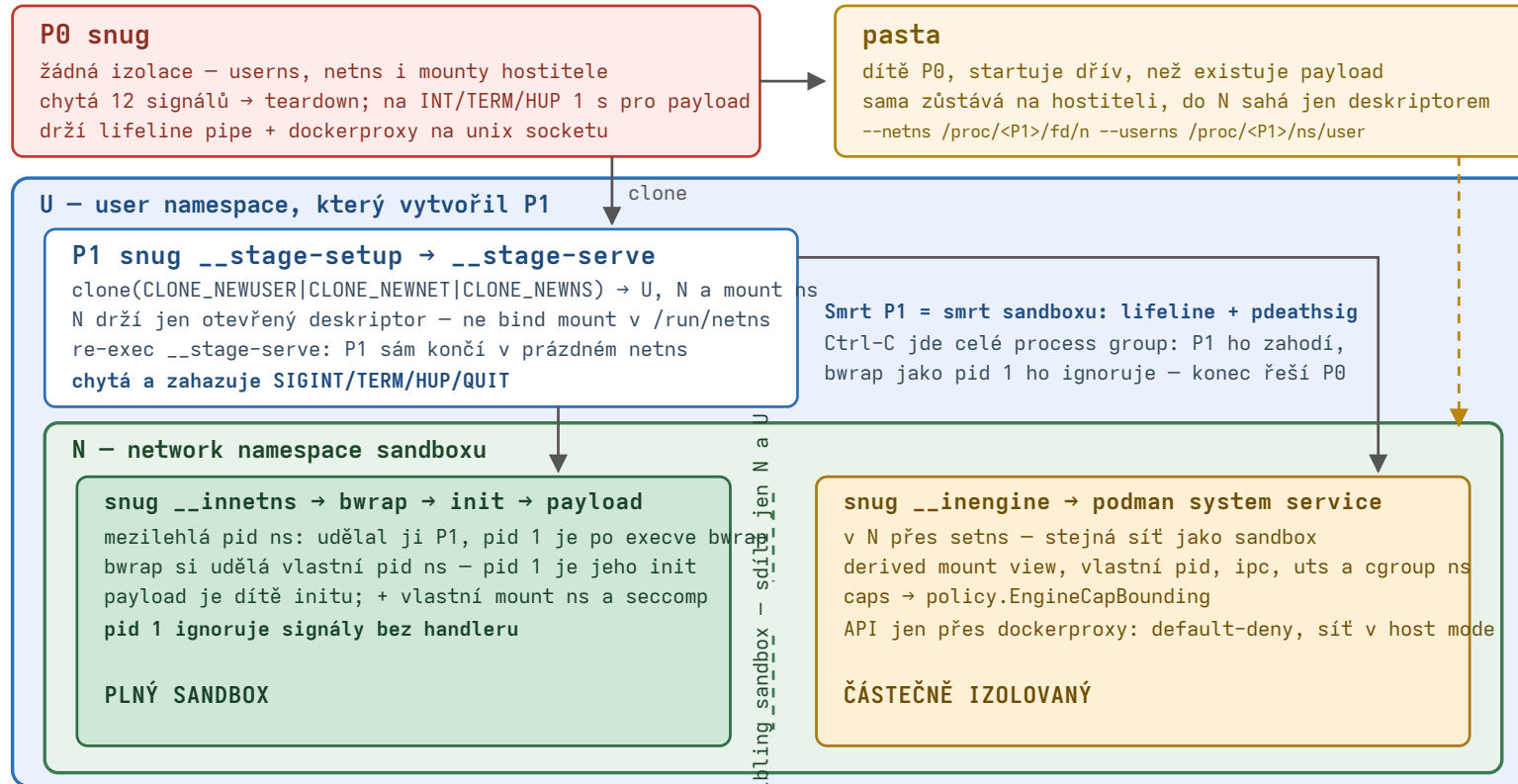
Your home directory:

```
/home/user
```

Both keep the exact path they have on your host. snug never relocates a path, because a tool that prints an absolute path inside should **print** one that means the same thing outside.

...

Jak to drží pohromadě



sibling sandbox: engine je sourozenec, ne nadřízený ani podřízený.

Kontejner dostane síť sandboxu, jeho -v prochází stejným filtrem jako mounty sandboxu.

Dosáhne na strom sandboxu plus grafty tohoto běhu – dál ne.

snug: stav projektu

- **alfa** - architektura a profilový model se usazují, ale věci se mohou měnit
- **Go a dvě přímé závislosti** - `go-toml/v2` a `golang.org/x/sys`. Statická binárka, nocgo, supply chain, která se vejde na řádek
- **59 tisíc řádků kódu a 106 tisíc řádků testů** - z 478 souborů je 331 testových
- **většinu kódu píše Claude** - 252 z 320 commitů ho má jako co-authora
- **na sandbox útočí vlastní agenti** - `redteam`, `sandbox-policy`, `sandbox-tester`, `host-bridge`. Co najdou, se zavře a přibude regresní test
- **chybí uživatelská dokumentace** - polovina README i `.claude/design/` je od LLM
- **dá se používat** - Na izolaci Claude Code. Nebo Javascriptu.

Q&A