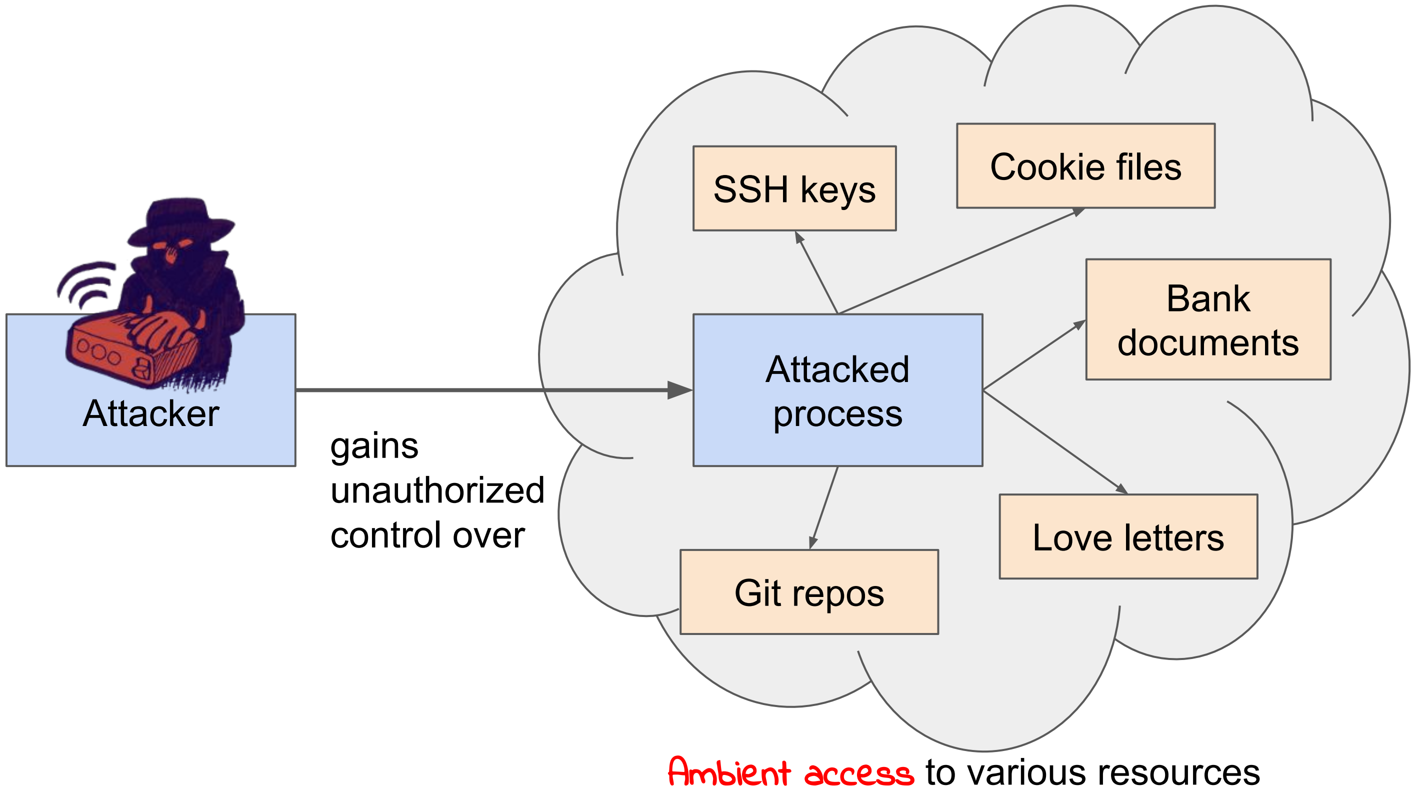
Go-Landlock

Günther Noack
blog.gnoack.org
2022-10-05

This slide deck: https://blog.gnoack.org/talks/go-landlock

High level overview of an attack

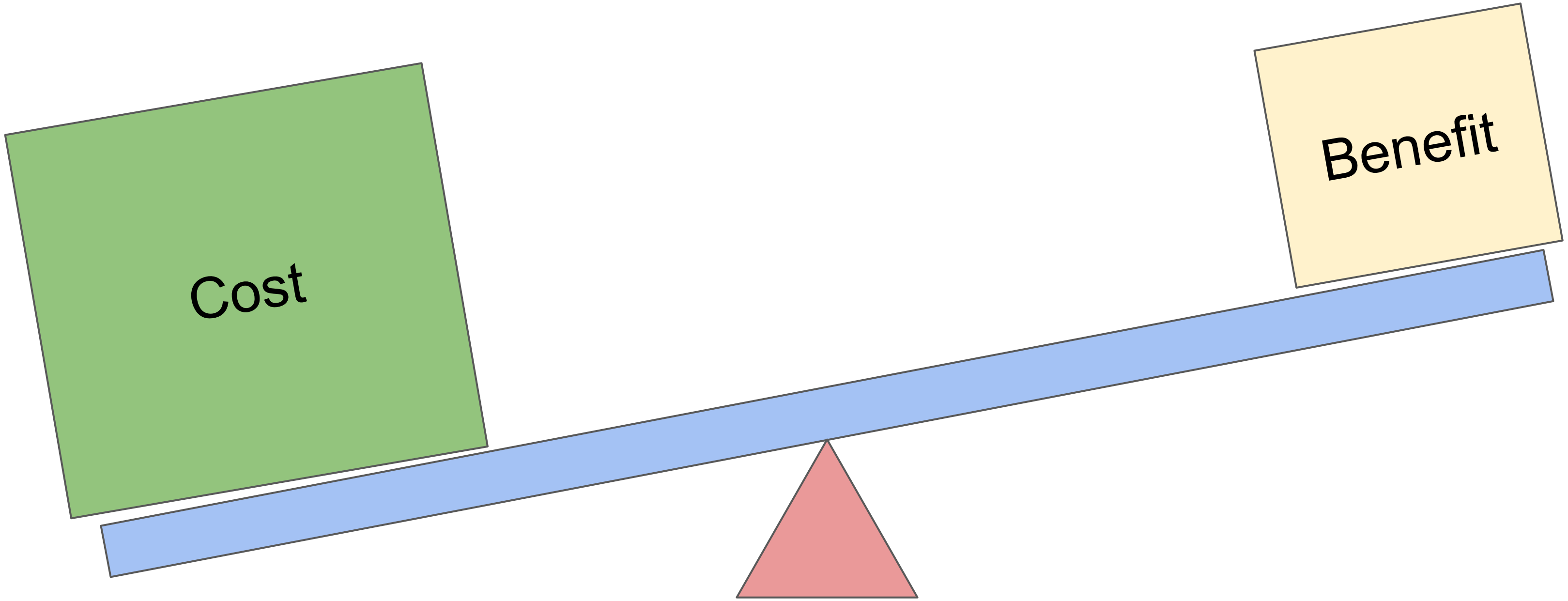


Let’s limit this ambient access!

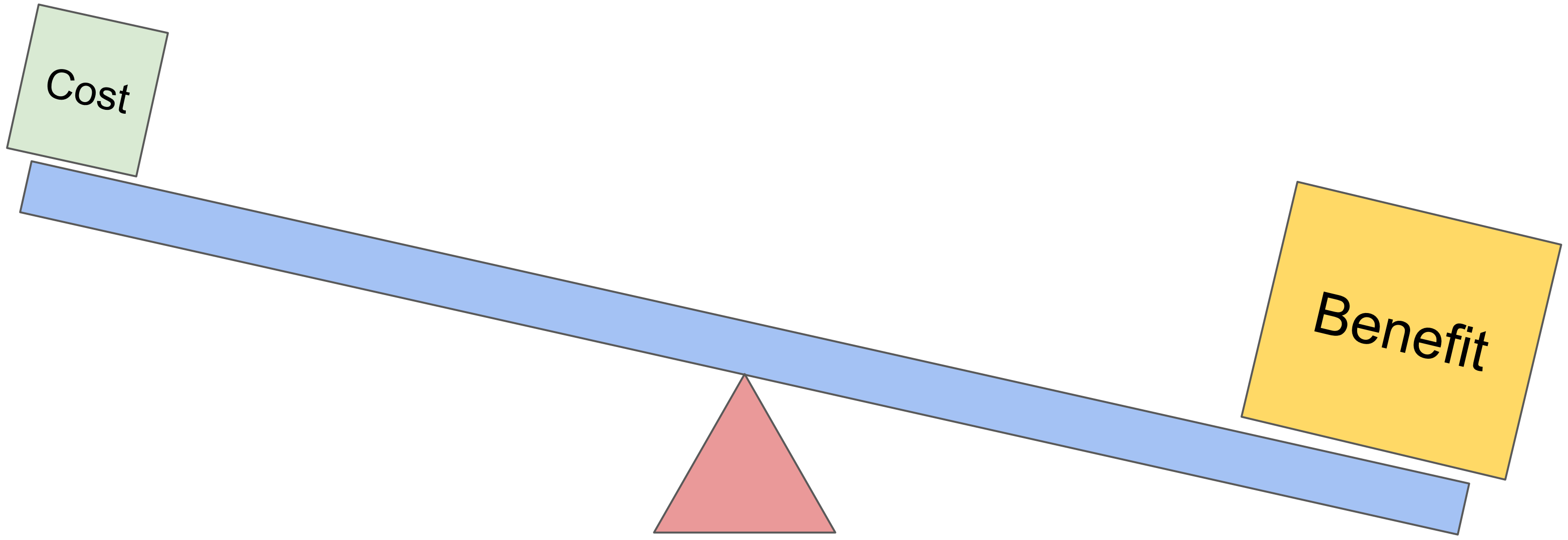
Show of hands!

- Who writes software that runs in a container? (docker, k8s, …)
- Who writes software that runs **outside** a container?
- Who has tried to sandbox their software?
- Why not?

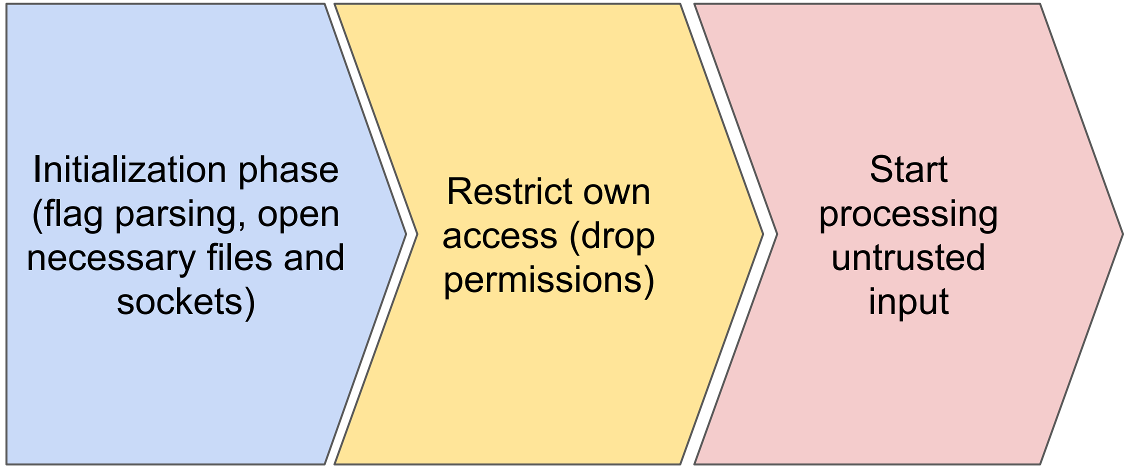
Limiting access is too hard with existing solutions!



Idea 1: Make it so simple that everyone can do it



Idea 2: Make it part of program initialization

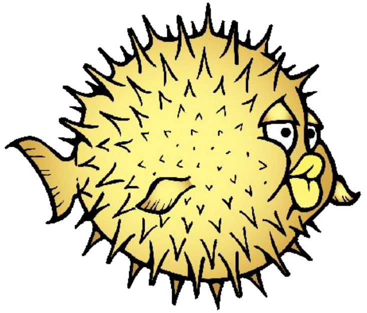

Initialization phase (flag parsing, open necessary files and sockets)
Restrict own access (drop permissions)
Start processing untrusted input

These ideas are not new

- OpenBSD: pledge() and unveil()

```
int pledge(const char *promises, const char *execpromises);
int unveil(const char *path, const char *permissions);
```

  Very lightweight to use from C, a lot of OpenBSD programs are “pledged”



- FreeBSD: Capsicum
  - A more full-fledged capability-passing security model
  - Removes all access to global namespaces

Unprivileged sandboxing on Linux

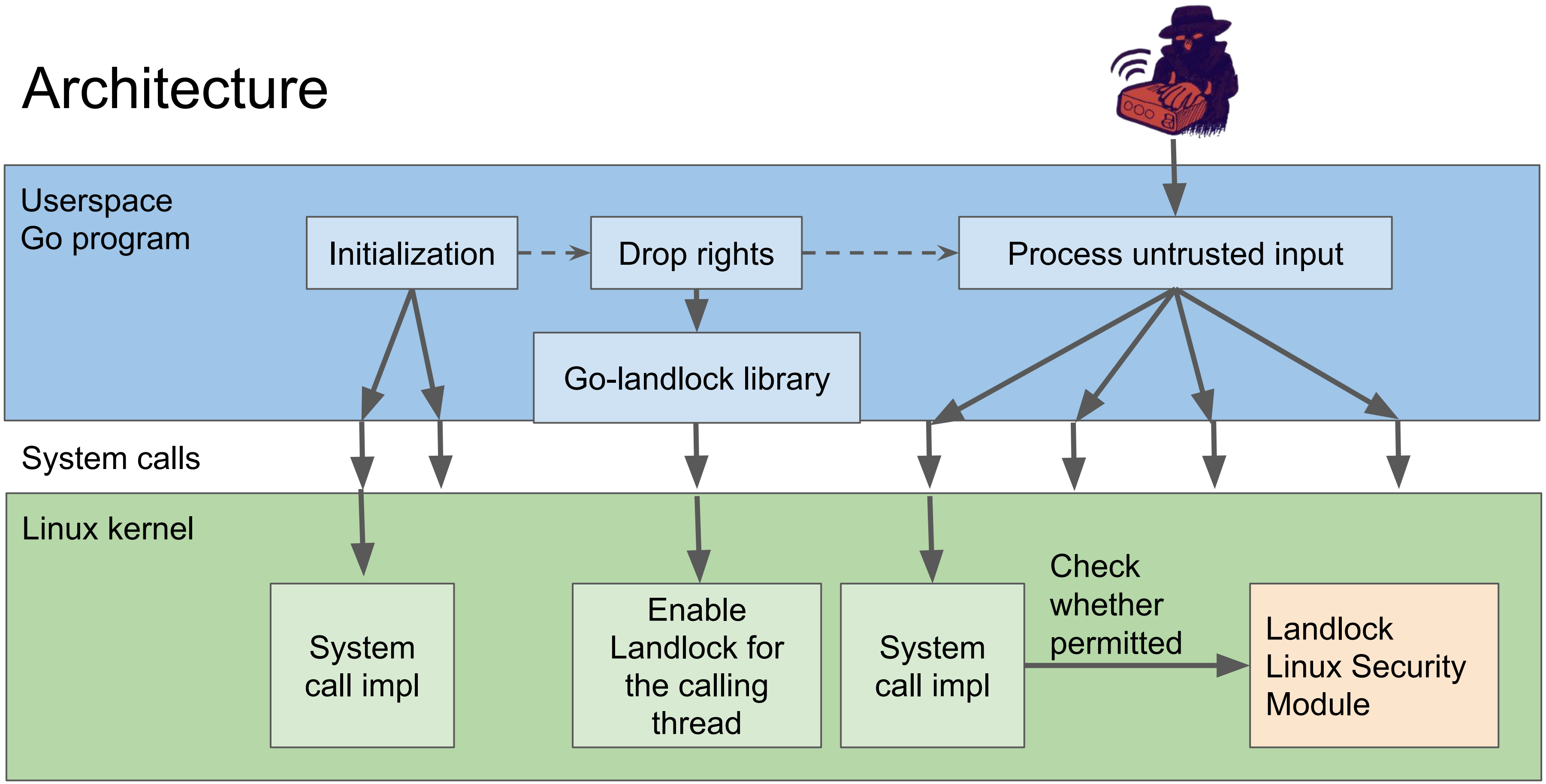
…is otherwise very hard to use

- Seccomp-BPF: System call filter in bytecode language
- User namespaces + Mount namespaces and other namespaces

(there are more detailed slides on these at the end, if needed)

How to use Go-Landlock

Architecture

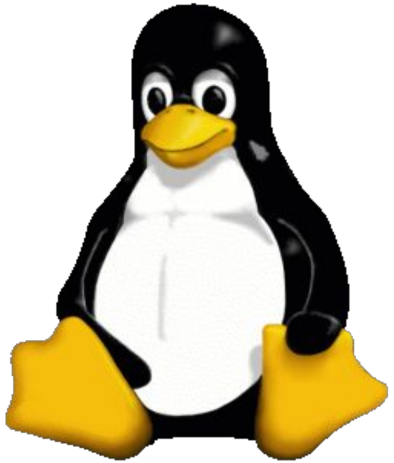


Step 1: Make sure your Linux kernel supports Landlock

- Needs to be (a) compiled into kernel and (b) enabled at boot time with lsm=landlock boot parameter (or by default with CONFIG_LSM) (source)
- Check whether you already have it enabled:

```
gnoack:~$ cat /sys/kernel/security/lsm
Capability,landlock,lockdown,yama,bpf
```

- Now supported by default in:
  - Alpine Linux
  - Arch Linux
  - chromeOS (including for Linux 5.10)
  - Debian Sid (testing)
  - Fedora 35
  - Ubuntu 20.04 LTS (source)



Step 2: State what file accesses you are going to do!

Use the highest Landlock ABI version you can, increase it opportunistically

Use the best set of Landlock features available on the current kernel

```
err := landlock.V2.BestEffort().RestrictPaths(
    landlock.RODirs("/usr", "/bin"),
    landlock.RWDirs("/tmp"),
)
```

Files we need to read*

Files we need to write*

* access can be made more granular if required

Example: Image converter

```
func main() {
    if err := landlock.V2.BestEffort().RestrictPaths(); err != nil {
        log.Fatal("Could not enable Landlock:", err)
    }

    imgData, _, err := image.Decode(os.Stdin)
    if err != nil {
        log.Fatal("Could not read input:", err)
    }

    if err := png.Encode(os.Stdout, imgData); err != nil {
        log.Fatal("Could not write output:", err)
    }
}
```

Drop access rights

Process untrusted input

https://github.com/landlock-lsm/go-landlock/blob/main/examples/convert/main.go

Example: Wiki software (simplified)

```
func main() {
    flag.Parse()

    d := diskv.New(diskv.Options{BasePath: *storeDir})
    http.Handle("/", &ukuleleweb.PageHandler{MainPage: *mainPage, D: d})

    s := http.Server{}
    l, err := net.Listen(*listenNet, *listenAddr)
    if err != nil { log.Fatalf("net.Listen: %v", err) }

    err = landlock.V2.BestEffort().RestrictPaths(
        landlock.RWDirs(*storeDir),
    )
    if err != nil { log.Fatalf("Landlock: %v", err) }

    err = s.Serve(l)
    if err != nil { log.Printf("http.ListenAndServe: %v", err) }
}
```

Unix Domain Socket!

Program initialization

Drop access rights

Process untrusted input

https://github.com/gnoack/ukuleleweb/blob/main/cmd/ukuleleweb/main.go

Example: Play with the go-landlock example tool

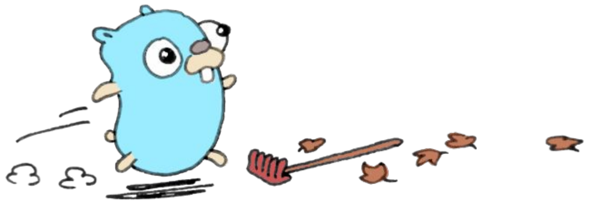
```
gnoack:~$ go install github.com/landlock-lsm/go-landlock/cmd/landlock-restrict@latest
gnoack:~$ export HOME=$(mktemp --directory -t tmphome-XXXXXXX)
gnoack:/home/gnoack$ export TMPDIR=$HOME/.localtmp
gnoack:/home/gnoack$ mkdir -p $TMPDIR
gnoack:/home/gnoack$ cd
gnoack:~$ landlock-restrict -ro /usr /lib /etc -rw "${HOME}" /dev -- /bin/bash
[gnoack@nuc ~]$ ls
[gnoack@nuc ~]$ pwd
/tmp/tmphome-zMtxO01
[gnoack@nuc ~]$ id
uid=1000(gnoack) gid=1000(gnoack) groups=1000(gnoack),962(docker)
[gnoack@nuc ~]$ ls ..
ls: cannot open directory '..': Permission denied
[gnoack@nuc ~]$
```

Current Limitations

Current limitations

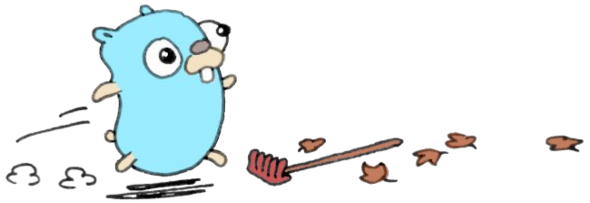
Some small things that Landlocked processes can never do:

- No manipulation of FS topology (i.e. mounting, pivot_root)
- NO_NEW_PRIVS flag: (i.e. executing suid root binaries)
- Restricted use of ptrace() (debugging other processes)

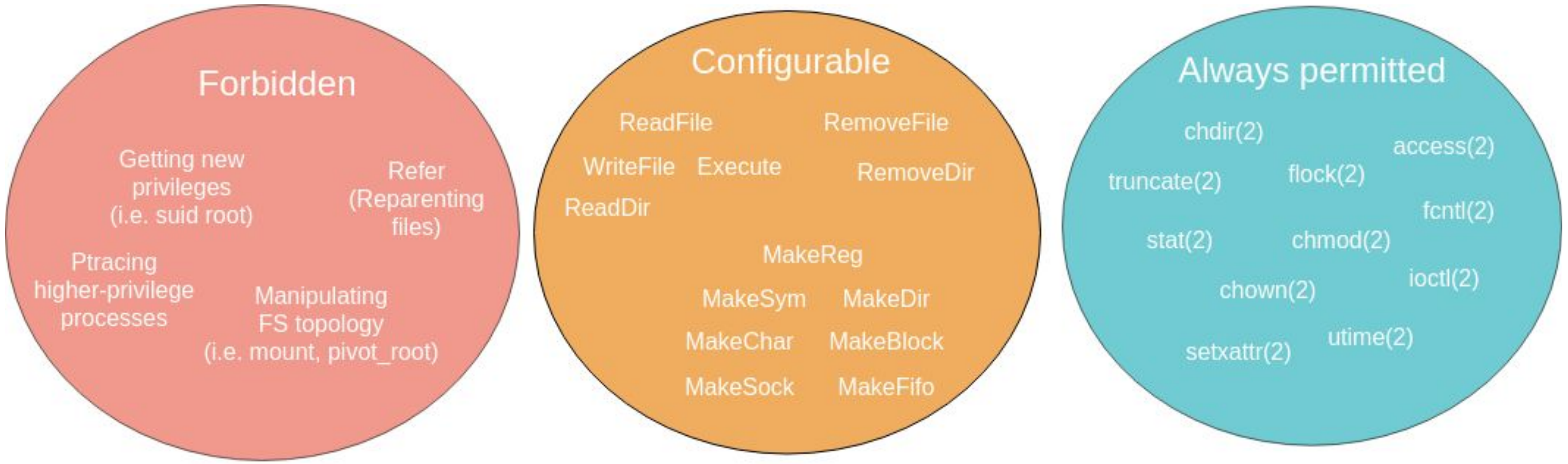


Current Limitations

- Landlock is **in development**.
- Is not able to restrict all file operations yet
- But it’s already limiting the most common ones :)

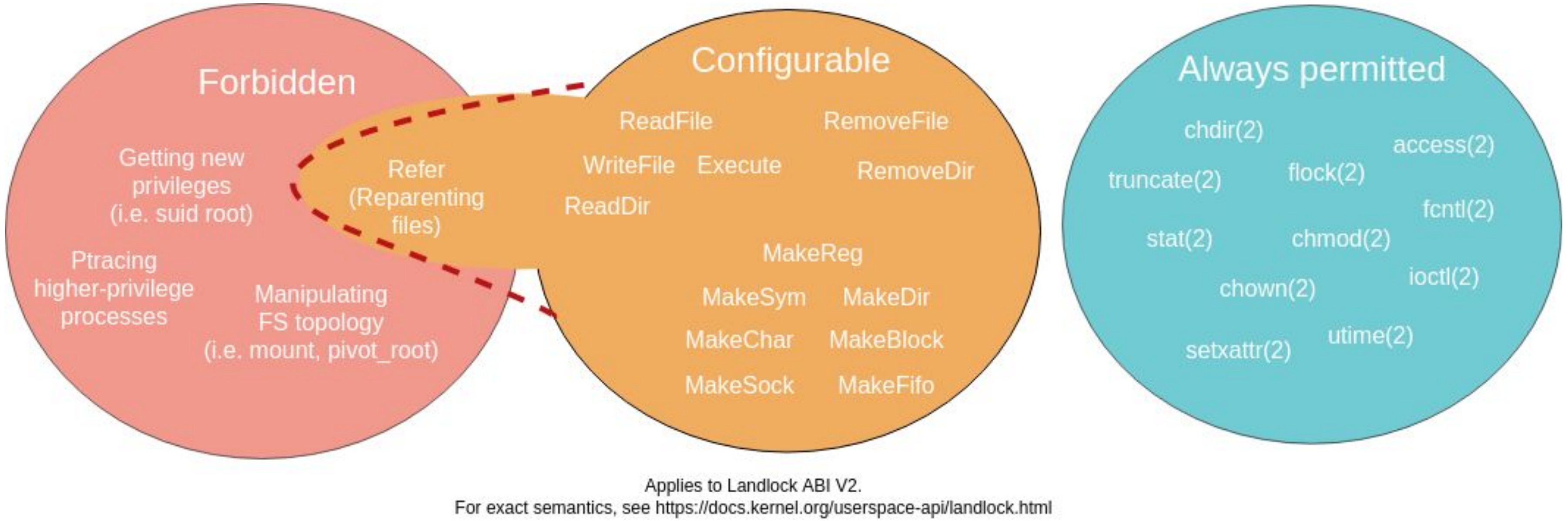


What is restrictable? (V1)



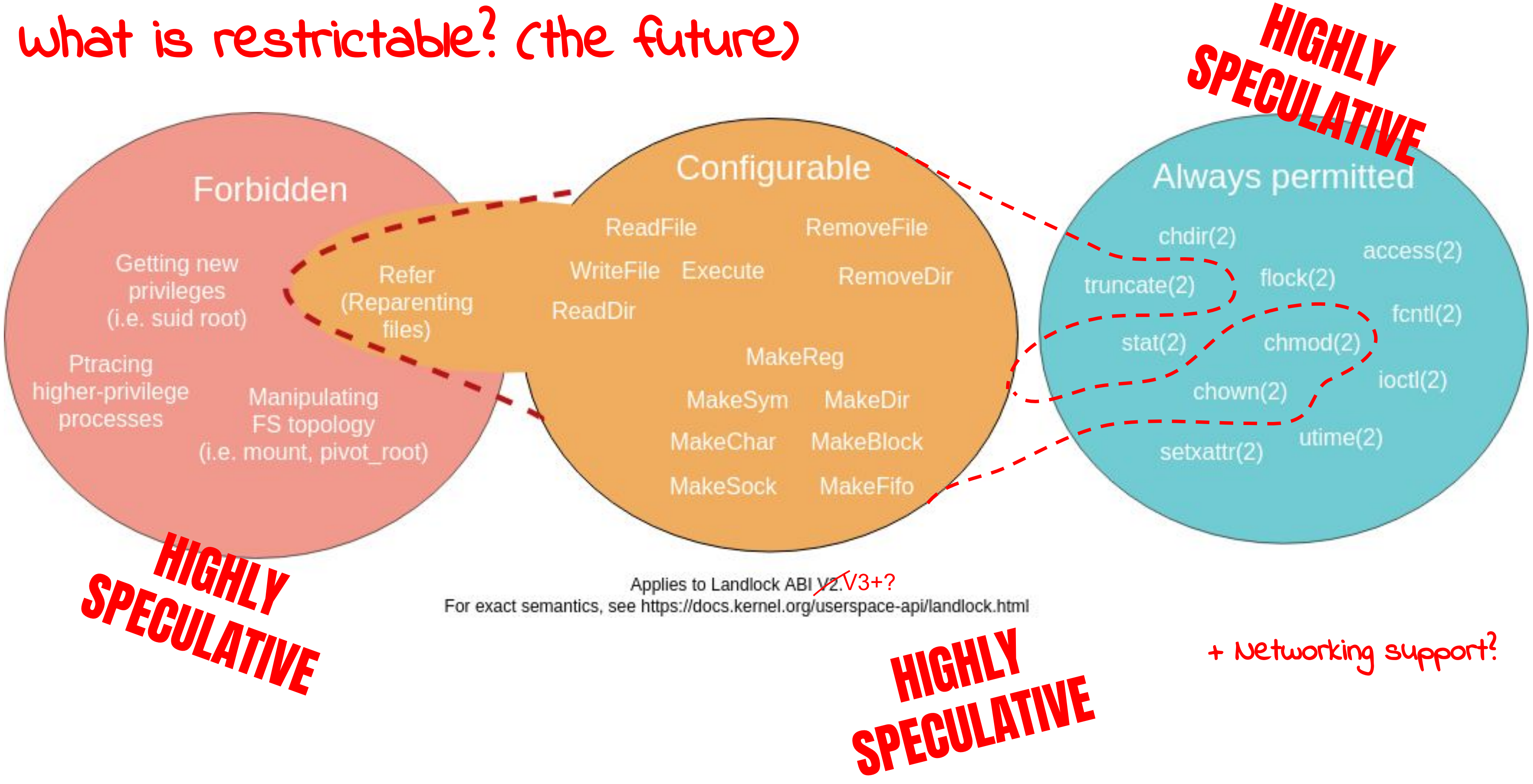
Applies to Landlock ABI V1.
For exact semantics, see https://docs.kernel.org/userspace-api/landlock.html

What is restrictable? (V2)


Forbidden
Getting new privileges (i.e. suid root)
Ptracing higher-privilege processes
Manipulating FS topology (i.e. mount, pivot_root)
Refer (Reparenting files)
Configurable
ReadFile
RemoveFile
WriteFile
Execute
RemoveDir
ReadDir
MakeReg
MakeSym
MakeDir
MakeChar
MakeBlock
MakeSock
MakeFifo
Always permitted
chdir(2)
access(2)
truncate(2)
flock(2)
fcntl(2)
stat(2)
chmod(2)
ioctl(2)
chown(2)
setxattr(2)
utime(2)
Applies to Landlock ABI V2.
For exact semantics, see https://docs.kernel.org/userspace-api/landlock.html

(also compare https://docs.google.com/document/d/1SkFpl_Xxyl4E6G2uYIlzL0gY2PFo-NI8ikblLvnpvlU/edit#)

what is restrictable? (the future)



Applies to Landlock ABI ~~V2~~.V3+?
For exact semantics, see https://docs.kernel.org/userspace-api/landlock.html

+ Networking support?

Key Point

Please try it out!

```
err := landlock.V2.BestEffort().RestrictPaths(
    landlock.RODirs("/usr", "/bin"),
    landlock.RWDirs("/tmp"),
)
```

I would ❤️ to hear your feedback

Landlock mailing list:

- https://lore.kernel.org/landlock/
- Subscribe: landlock+subscribe@lists.linux.dev

Or to my own email:

- gnoack3000@gmail.com

```
PGP: 7F02 BDCC 6157 6E11 1A87
     9BD1 1C62 9E5A F9E8 CDA1
```

Thank you!

Links

Go-Landlock:

- Source: https://github.com/landlock-lsm/go-landlock
- Docs: https://pkg.go.dev/github.com/landlock-lsm/go-landlock/landlock

Landlock Linux Security Module:

- https://landlock.io/
- Kernel docs: https://docs.kernel.org/userspace-api/landlock.html

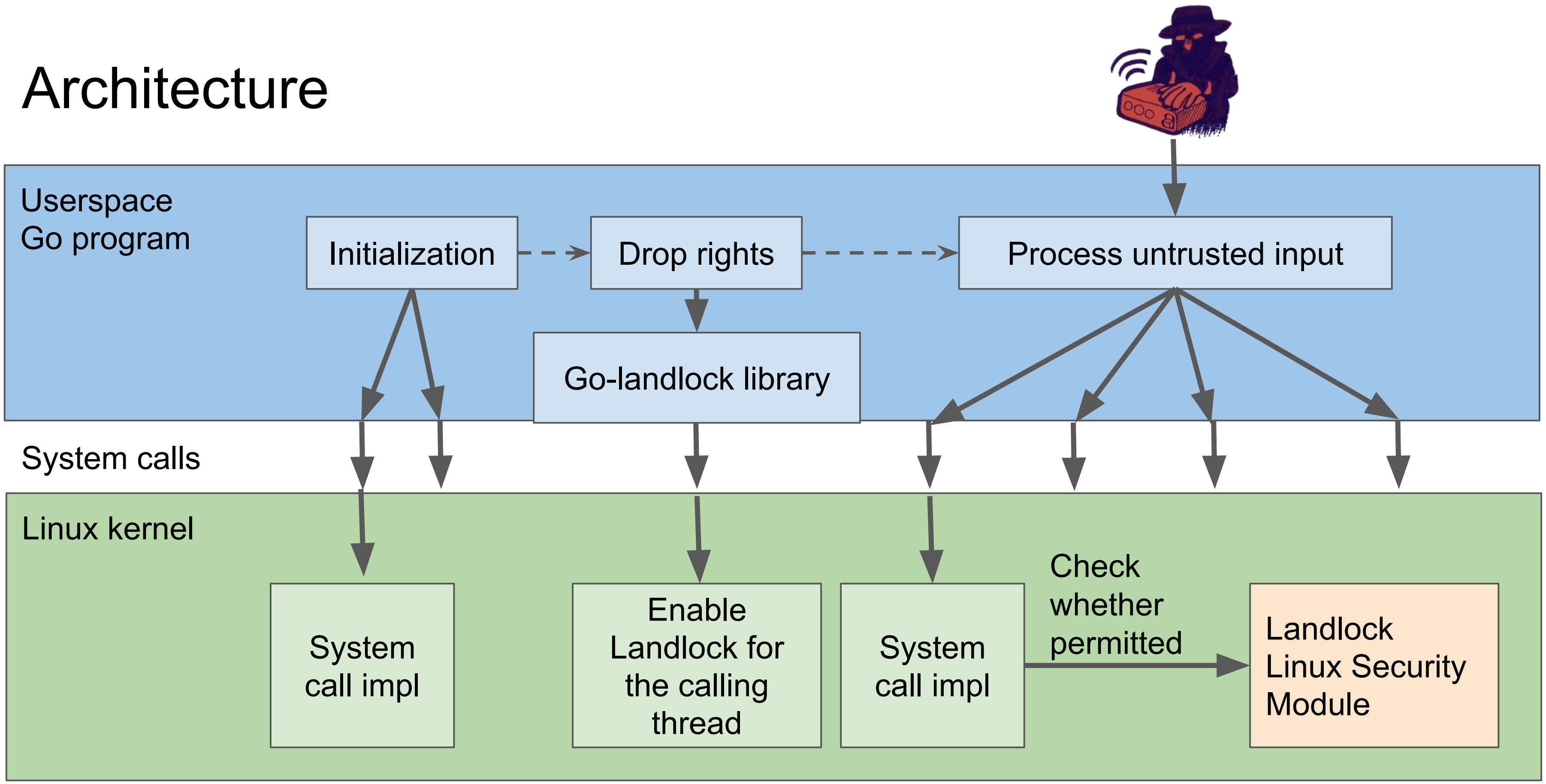
This talk: https://blog.gnoack.org/talks/go-landlock

Questions

Bonus Slides

Go-Landlock Implementation

Architecture



How does Landlock get enabled?

- **Create a Landlock ruleset** file descriptor
- For each path we want to use:
  - **Open path** with O_PATH
  - **Add path and its allowed access rights to landlock ruleset**
- **Enforce Landlock ruleset** on the current thread

Pop quiz: How many Goroutines are running here?

```
func main() {

    err := landlock.V2.BestEffort().RestrictPaths()
    // …
    callSomeFunc()
}
```

… and how many OS threads?

Answer: **Too many!**
The Go runtime already starts goroutines before main() 😱

syscall.AllThreadsSyscall to the rescue

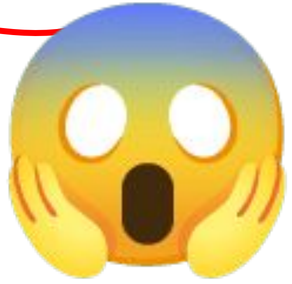
```
syscall.AllThreadsSyscall(
    SYS_LANDLOCK_RESTRICT_SELF,
    uintptr(rulesetFd), uintptr(flags), 0)
```

A helper exposed by the runtime:

- Invokes a system call on each OS Thread managed by the runtime
- Expects that all syscalls return the same error

Works for Go! \o/

But not for cgo



Libpsx to the rescue

- Part of libcap project
- Some syscalls are just thread-only

So…

- Learn about identity of all threads: intercept pthreads with a linker hack
- Invoke syscall on all OS threads:
    - Register a special signal handler under an unused(!) signal number for all threads
    - Signal all threads, so that they'll execute the syscall from that signal handler
    - Collect results from threads through global variable

https://sites.google.com/site/fullycapable/who-ordered-libpsx explains it in detail

The upside: This sounds more horrible than it is

- The other main user of this implementation technique:

Glibc

- You are already relying on this approach today…

Testing learnings…

- Needed to create subprocesses to run the actual tests
  - Landlock policies do not play nicely with the test framework
- It pays off to run Go tests in qemu under different kernels
  - florianl’s bluebox framework has helped to get this working

Other Linux Sandboxing technology

Seccomp-BPF

- Unprivileged :)
- Install a “firewall” for system calls to be used later on
  - System call filter based on syscall number and (register) arguments
  - Requires to write BPF bytecode or to use larger libraries
- The list of system calls is not static
  - Differs between architectures
  - Differs between kernel versions
  - As of 5.19, 363 syscalls for x86_64, 352 syscalls for x86
  - Difficult to maintain an up to date list as a side project
  - Libraries do not usually give guarantees about the system calls they use
- Users: Chromium, OpenSSH, Firefox, Tor, some container software…
- https://blog.gnoack.org/post/pledge-on-linux/

Mount namespaces

- `unshare(CLONE_NEWNS)`
- Requires CAP_SYS_ADMIN (you need to be root-ish)
- You can acquire CAP_SYS_ADMIN with `clone(..., CLONE_NEWUSER)`
  - `Can only be done at program execution boundary`
- Process environment will be different than you'd expect, it's not very transparent to the program being sandboxed.

Same goes for most other namespaces (network, pid, ipc, …)

AppArmor, SELinux, SMACK, TOMOYO

- Are also Linux Security Modules
- Sandboxing “from the outside” (more coarse)
- System administrator defines execution policies
- Inconsistent availability. Ubuntu uses AppArmor, RedHat uses SELinux.
- Enabling both AppArmor and SELinux in parallel (“LSM stacking”) is work in progress

Various command line tools, firejail and friends

- Usually require root
  - Escalating privileges to drop privileges…?
  - Increase of TCB
- These build on combinations of various namespaces and more complicated seccomp mechanisms