pledge

and why you should use it

The Long Con 2023

Hello + basic disclaimer

I’m Rob Keizer.

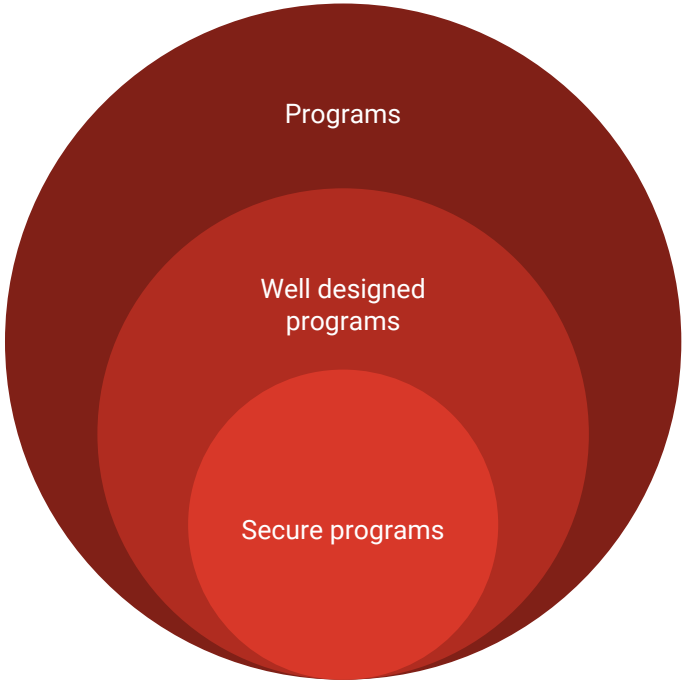
I’m speaking as myself.

This is an introduction to pledge.

What we’re covering

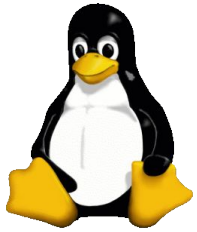
- Why does pledge exist?
- Surely this existed before…
- Recent(ish) history and why I’m talking about it.
- Basic usage of pledge.
- That’s it. Seriously.

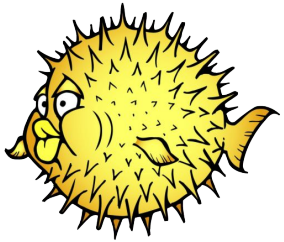
Why does pledge exist?

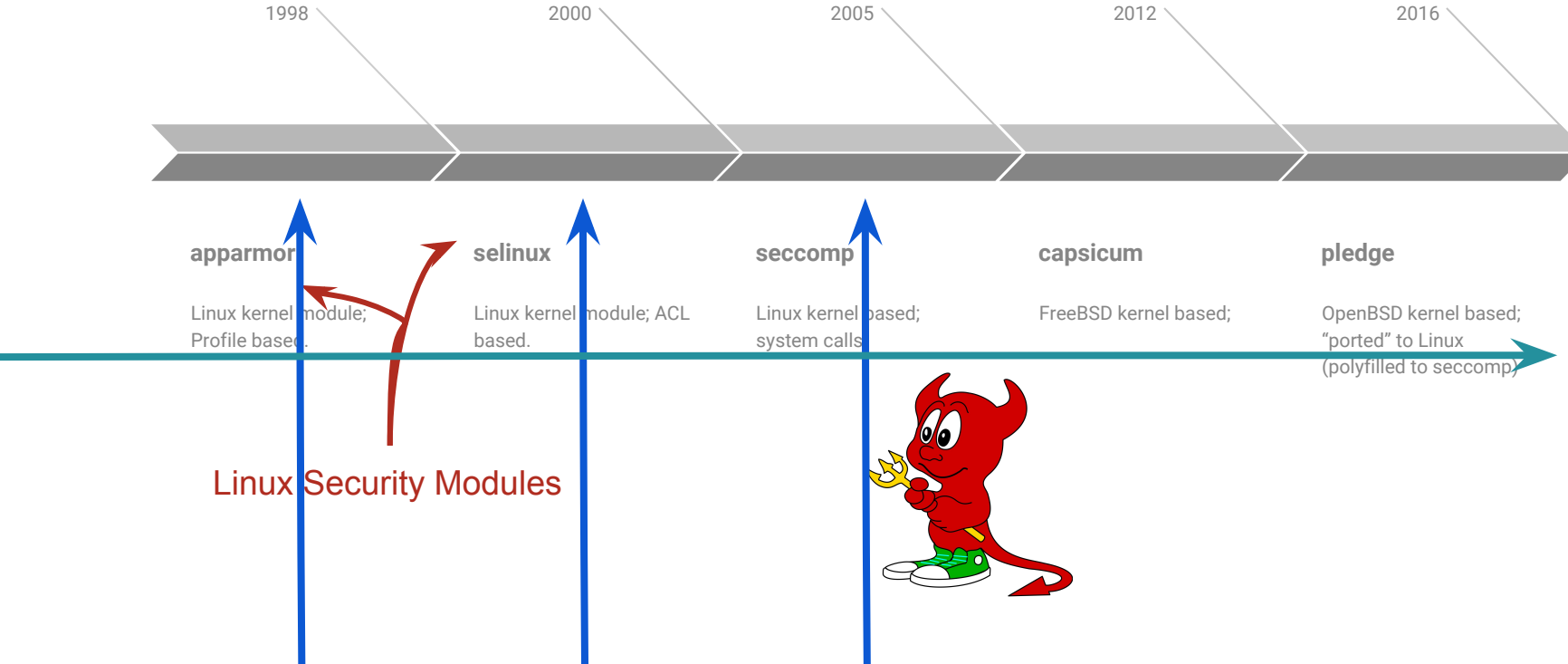


- Programs do a bunch of stuff
- Most security issues are in programs
- Programs know what they *should* do
- Operating systems provide system calls
- Why not limit the system calls?

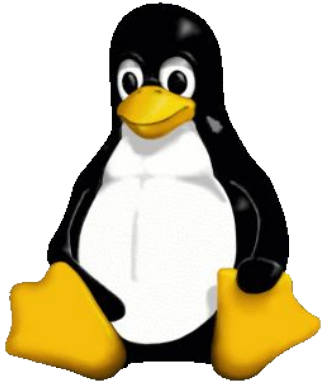
This seems familiar …



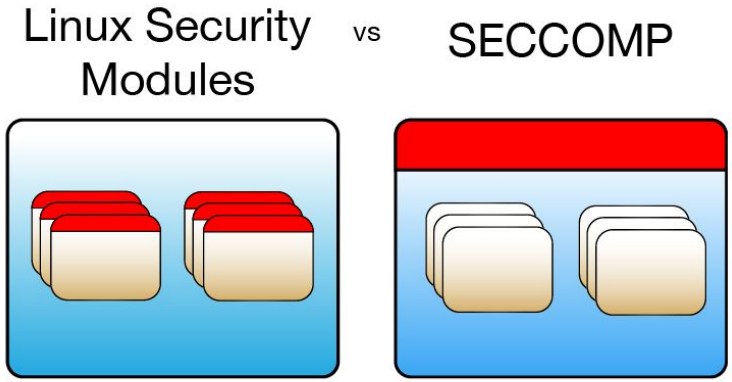




LSMs vs seccomp



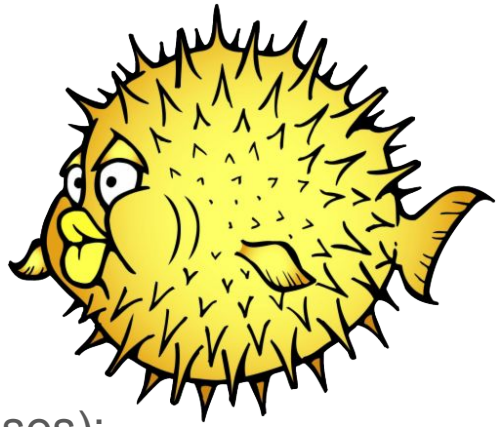
User Program



- Similar but different
- LSMs hook on kernel objects
- seccomp limits system calls

- Apparmor/SELinux use LSM
- OS applies LSMs
- Apps use seccomp

Back to pledge



- Native to OpenBSD
- Similar(ish) to seccomp
  - Generalization; Don’t hate me.
- int pledge(const char *promises, const char *execpromises);
  - promises are groups of system calls; “stdio”, “rpath”, “wpath”, “inet”, “proc”, “exec”, etc…
  - execpromises are inherited by children
- Sane abstractions that are similar to FreeBSD capabilities

Recent(ish) history

- Polyfilled to Linux in 2022
  - SECCOMP BPF by Justine Tunney

Actually usable!

Promise to BPF

```
static const struct sock_filter kFilter[] = {
    /* L0*/ BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, syscall, 0, 14 - 1),
    /* L1*/ BPF_STMT(BPF_LD | BPF_W | BPF_ABS, OFF(args[0])),
    /* L2*/ BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, 2, 4 - 3, 0),
    /* L3*/ BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, 10, 0, 13 - 4),
    /* L4*/ BPF_STMT(BPF_LD | BPF_W | BPF_ABS, OFF(args[1])),
    /* L5*/ BPF_STMT(BPF_ALU | BPF_AND | BPF_K, ~0x80800),
    /* L6*/ BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, 1, 8 - 7, 0),
    /* L7*/ BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, 2, 0, 13 - 8),
    /* L8*/ BPF_STMT(BPF_LD | BPF_W | BPF_ABS, OFF(args[2])),
    /* L9*/ BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, 0, 12 - 10, 0),
    /*L10*/ BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, 6, 12 - 11, 0),
    /*L11*/ BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, 17, 0, 13 - 11),
    /*L12*/ BPF_STMT(BPF_RET | BPF_K, SECCOMP_RET_ALLOW),
    /*L13*/ BPF_STMT(BPF_LD | BPF_W | BPF_ABS, OFF(nr)),
    /*L14*/ /* next filter */
};
```

```
pledge("stdio rpath", 0);
```

Basic usage of pledge

openbsd

Rust bindings for OpenBSD's pledge(2) and unveil(2).

Usage

Pledge

Macro syntax

```
use openbsd::pledge;

pledge!("stdio rpath exec")?; // only make promises
pledge!(_, "stdio rpath")?; // only make execpromises
pledge!("stdio", "stdio")?; // make both

assert!(pledge!("wpath").is_err()); // cannot increase permissions
```

Basic usage of pledge

Usage

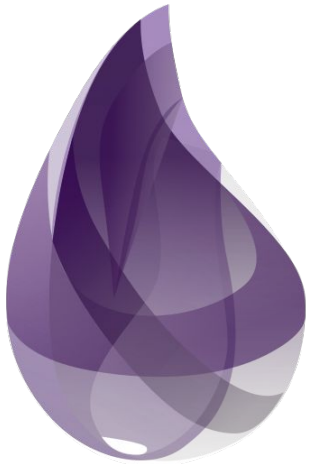
```
const pledge = require('openbsd-pledge')
const fs = require('fs')

pledge('stdio')
const fd = fs.openSync('test.txt', 'r', 0o555) // SIGABRT here
```

Basic usage of pledge

```
describe "pledge_promises/1" do
  test "Returns {:error, :einval} for invalid input" do
    assert {:error, :einval} = Pledge.pledge_promises("invalid_input")
  end

  test "Returns :ok for valid input" do
    assert :ok = Pledge.pledge_promises("stdio rpath wpath cpath vminfo ps error")
  end
end
```



Basic usage of pledge

```
_pledge = None
try:
    _pledge = ctypes.CDLL(None, use_errno=True).pledge
    _pledge.restype = ctypes.c_int
    _pledge.argtypes = ctypes.c_char_p, ctypes.c_char_p
except Exception:
    _pledge = None
```

```
def pledge(promises: Optional[str], execpromises: Optional[str]):
    if not _pledge:
        return  # unimplemented

    r = _pledge(None if promises is None else promises.encode(),
                None if execpromises is None else execpromises.encode())
    if r == -1:
        errno = ctypes.get_errno()
        raise OSError(errno, os.strerror(errno))
```

That’s it

questions and/or heckles