

Writing Exploit-Resistant Code with OpenBSD

Lawrence Teo

lteo@openbsd.org

@lteo

Slides and references at:

<https://lteo.net/carolinacon15>

A question

Innovation's Black Hole



**Security
vulnerabilities**

Image Credit:
EHT Collaboration
<https://www.eso.org/public/images/eso1907a/>

What is OpenBSD?

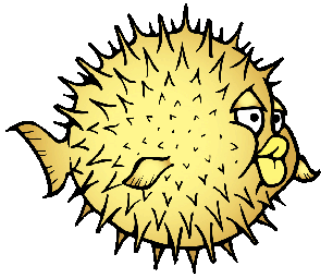
- Free, multi-platform UNIX-like operating system
- Founded by Theo de Raadt in 1995
- Secure by default
- A research operating system
- Two releases per year
- You're very likely using OpenBSD code everyday
 - OpenSSH
 - LibreSSL
 - tmux
 - More: openbsd.org/innovations.html
- Coolest mascot ever





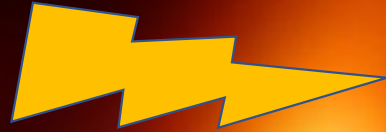
whoami

- OpenBSD developer since 2012
 - Primarily areas related to networking
 - PF, network stack, libpcap, tcpdump, etc.
 - Userland stuff, ports, man pages, etc
- Co-founder, Calyptix Security
 - Shipping thousands of OpenBSD-based firewalls from Charlotte since 2006!
- Ph.D. from UNC Charlotte (2006)
 - Research area: Info sharing for intrusion detection



***Open*BSD**

Auditing

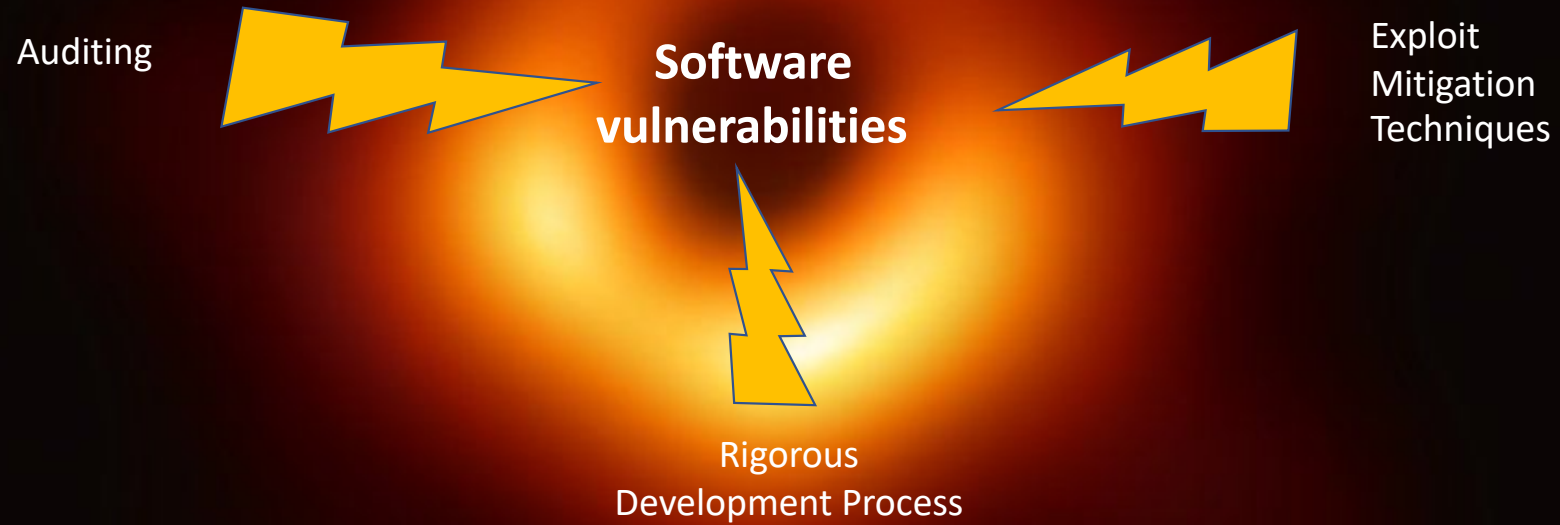


**Software
vulnerabilities**

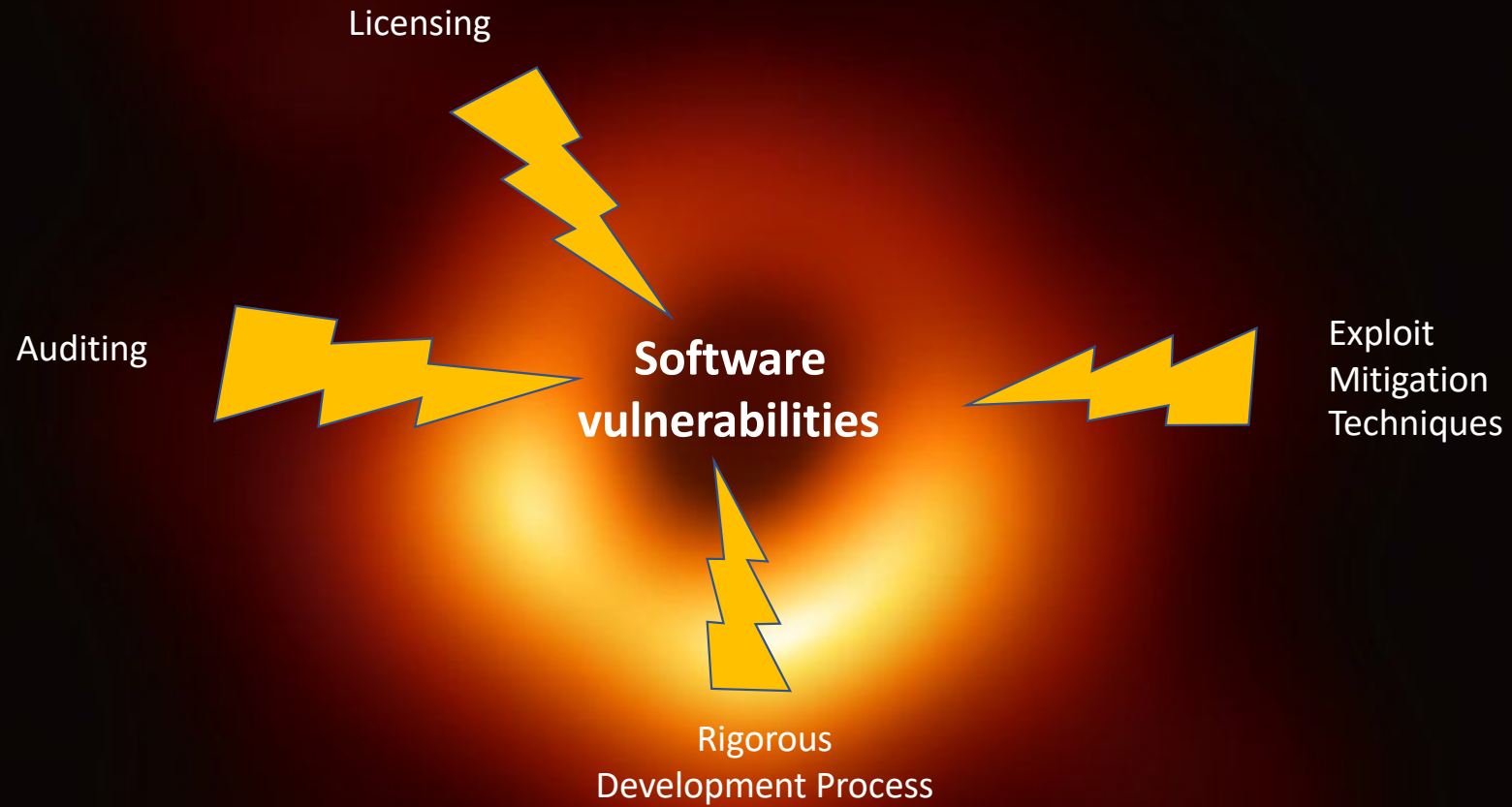
How OpenBSD attacks the software vulnerability problem (my view)



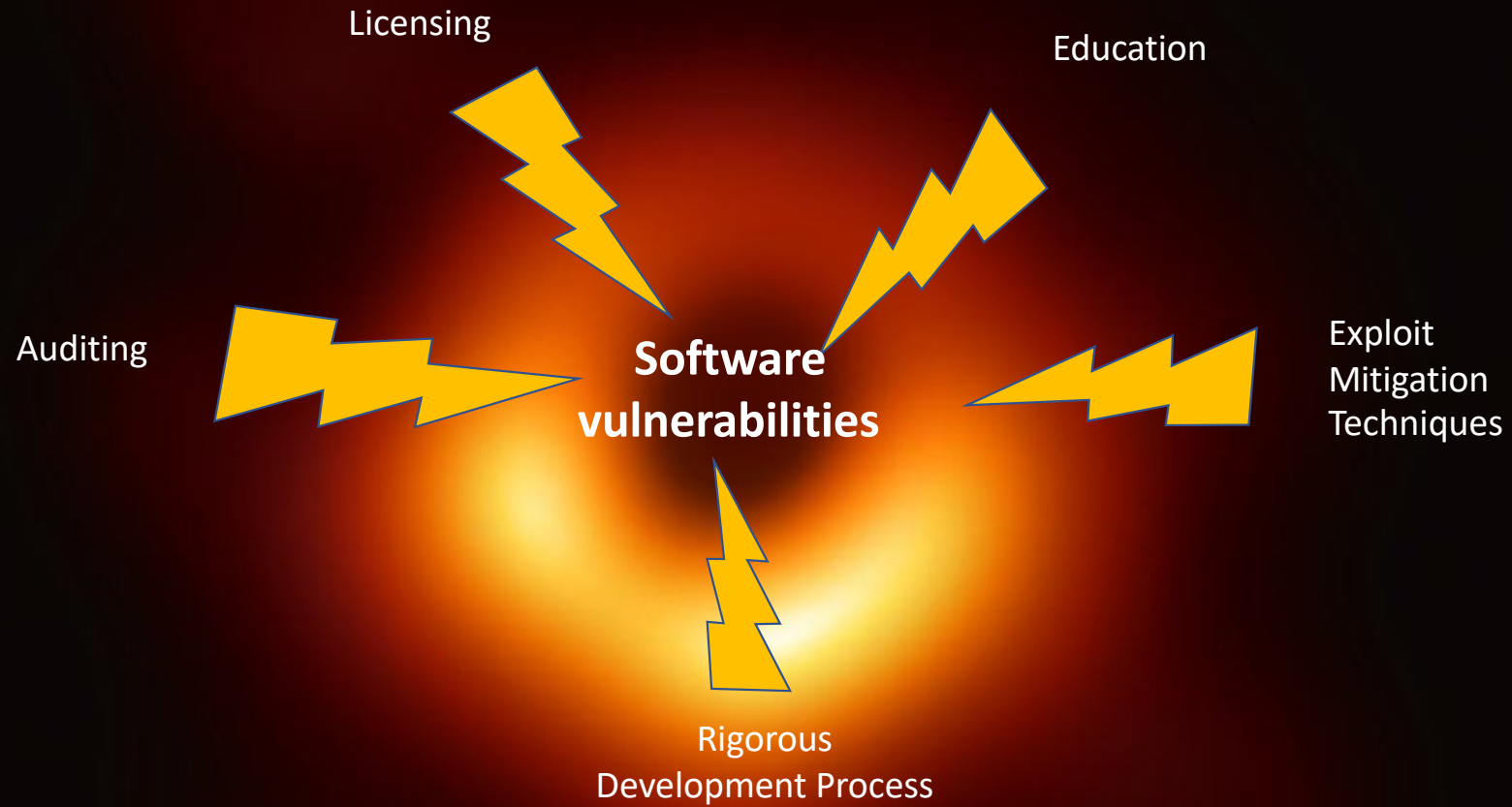
How OpenBSD attacks the software vulnerability problem (my view)



How OpenBSD attacks the software vulnerability problem (my view)

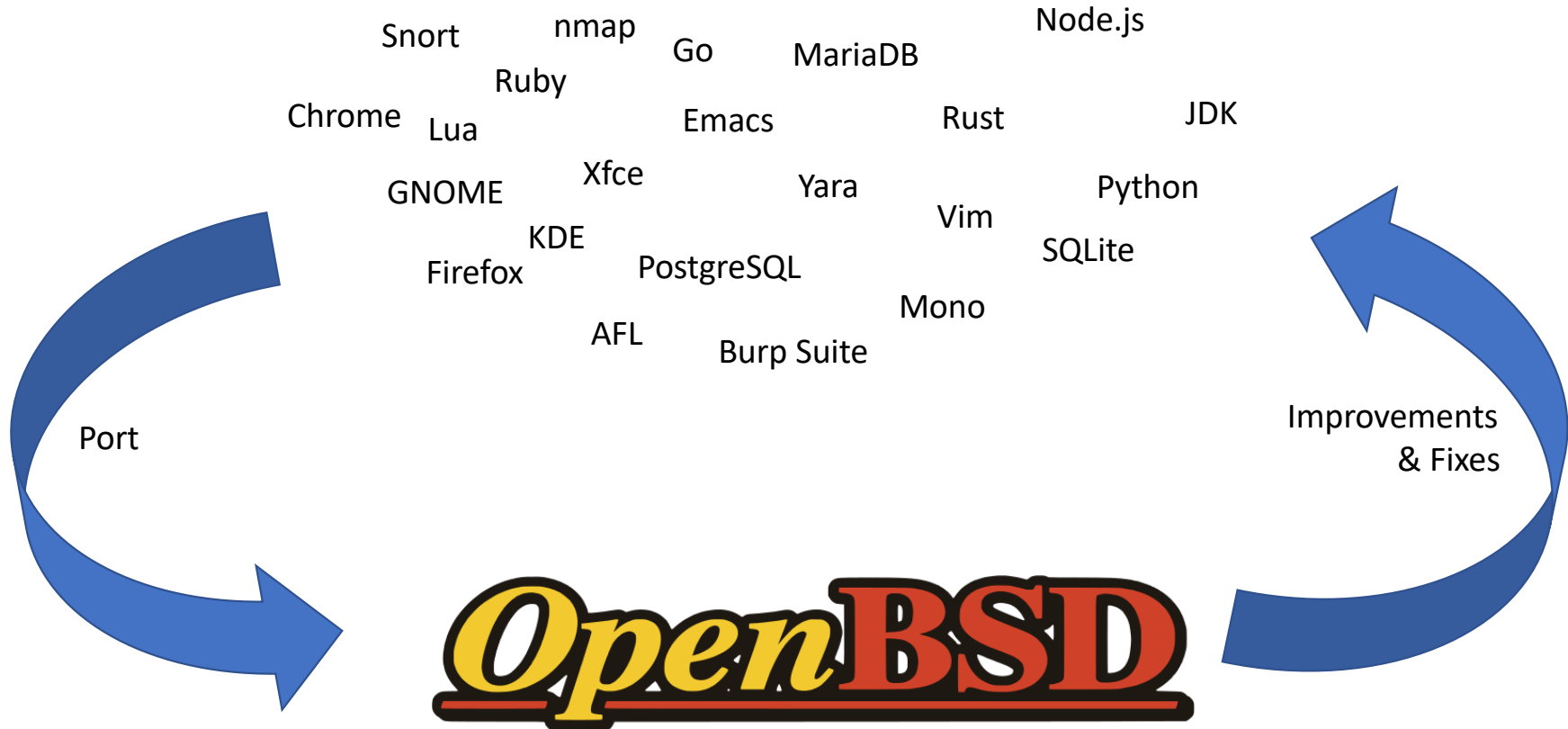


How OpenBSD attacks the software vulnerability problem (my view)



How OpenBSD attacks the software vulnerability problem (my view)

OpenBSD and upstream ports



OpenBSD: Not your typical Unix system

- Encrypted swap by default
- No loadable kernel modules
- No /proc
- Runs in secure mode by default (`kern.securelevel == 1`)
- Minimal daemons by default
- High-quality well-documented man pages
- Easy installer
- No systemd 😊
- Gobs of exploit mitigation techniques!

What is a mitigation?

“Mitigations are inexpensive tweaks which increase the difficulty of performing attack methods:

- low impact on normal operation
- huge impact during attack-scenario operation”

- Theo de Raadt

[“Mitigations and other real security features”](#), BSDTW 2017

- Attributes:
 - Pressures software to be more robust
 - Fail closed

Exploit Mitigation Techniques on OpenBSD

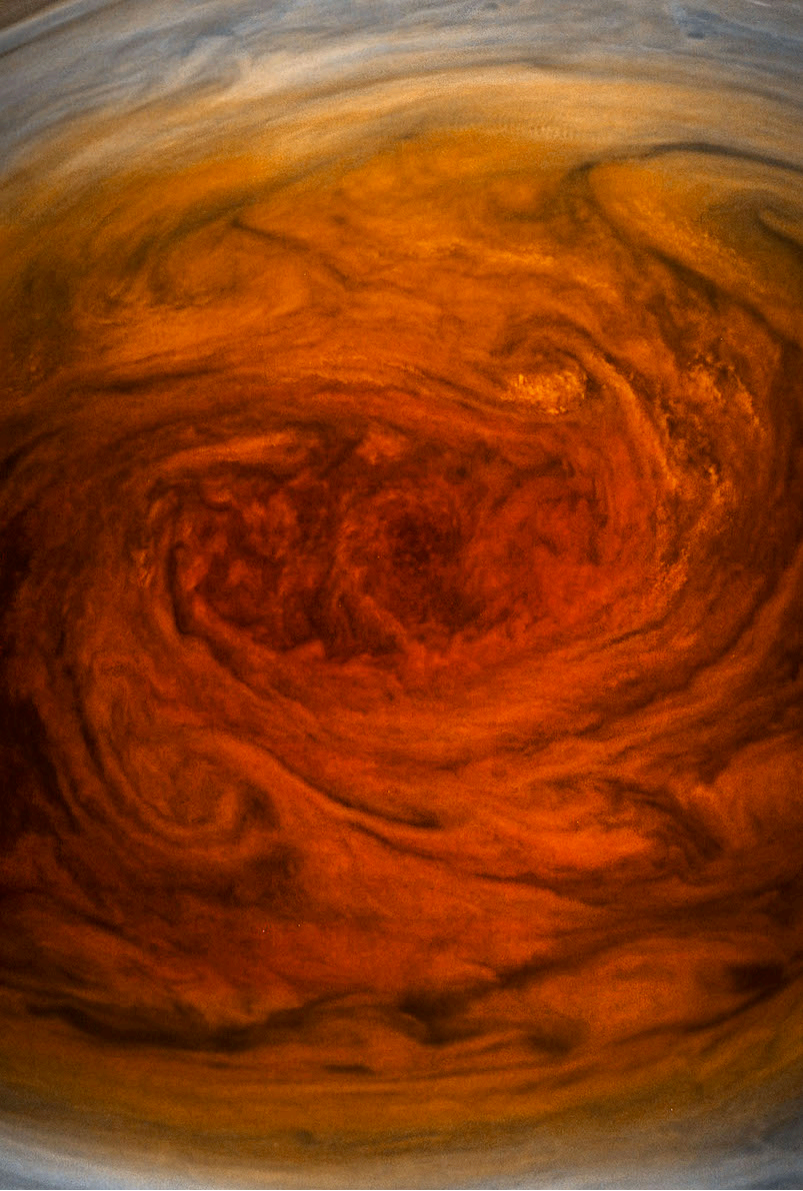
(non-exhaustive)

- Privilege separation
- Privilege dropping
- Stack protector
- W^X
- GOT and PLT protection
- ASLR
- Otto malloc (randomized malloc)
- PIE
- .openbsd.randomdata
- Stack protector per stack object
- Static-PIE
- SROP mitigation
- Library order randomization
- kbind() for W^X lazy binding
- RELRO
- fork+exec
- trapsleds
- KARL
- MAP_STACK
- ROP gadget removal
- RETGUARD
- pledge(2)
- unveil(2)
- ...

More at <https://www.openbsd.org/innovations.html> !



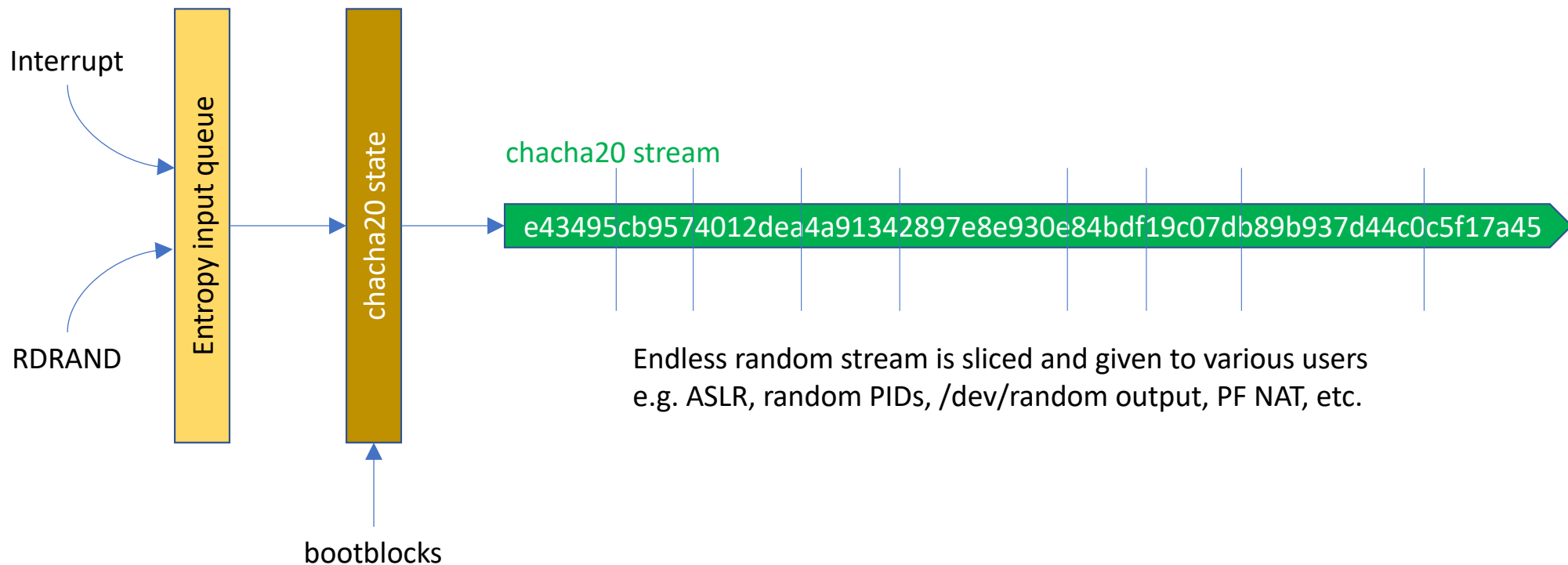
OpenBSD is a “hostile”
environment for code



A hostile environment

- OpenBSD challenges the assumptions that software makes
 - For both good and bad code
- When things fail, they crash -> GOOD
- Run your code on OpenBSD
 - Make sure your code can work in a hostile environment
- Can your code run on OpenBSD?
 - If behemoths like Chrome and Firefox can run on OpenBSD, your code likely can too!

Randomization in OpenBSD (simplified view)



Details at:

Theo de Raadt, "arc4random – randomization for all occasions", Hackfest 2014

<https://www.openbsd.org/papers/hackfest2014-arc4random>

Random PIDs

OpenBSD

```
obsd-hack01$ cat test.sh
#!/bin/sh
sleep 60
obsd-hack01$ sh test.sh &
[1] 11202
obsd-hack01$ sh test.sh &
[2] 56071
obsd-hack01$ sh test.sh &
[3] 12613
obsd-hack01$ pgrep -fl test.sh
12613 sh test.sh
56071 sh test.sh
11202 sh test.sh
```

macOS

```
macbook-air:~$ cat test.sh
#!/bin/sh
sleep 60
macbook-air:~$ sh test.sh &
[1] 1124
macbook-air:~$ sh test.sh &
[2] 1126
macbook-air:~$ sh test.sh &
[3] 1128
macbook-air:~$ pgrep -fl test.sh
1124 sh test.sh
1126 sh test.sh
1128 sh test.sh
```

Source ports

OpenBSD

```
22:20:52.014498 192.168.52.199.22294 > 172.217.12.100.80: S 605289938:605289938(0) win 16384 <mss
1460,nop,nop,sackOK,nop,wscale 6,nop,nop,timestamp 3876378313 0> (DF)

22:21:01.148513 192.168.52.199.12538 > 64.233.177.106.80: S 3877617101:3877617101(0) win 16384 <mss
1460,nop,nop,sackOK,nop,wscale 6,nop,nop,timestamp 2572651157 0> (DF)

22:21:18.923117 192.168.52.199.4112 > 184.25.123.162.80: S 755920089:755920089(0) win 16384 <mss
1460,nop,nop,sackOK,nop,wscale 6,nop,nop,timestamp 2718008402 0> (DF)

22:21:19.168733 192.168.52.199.5160 > 184.25.123.162.443: S 1720505599:1720505599(0) win 16384 <mss
1460,nop,nop,sackOK,nop,wscale 6,nop,nop,timestamp 1343246526 0> (DF)

22:21:26.692920 192.168.52.199.6455 > 52.205.209.212.80: S 3061487688:3061487688(0) win 16384 <mss
1460,nop,nop,sackOK,nop,wscale 6,nop,nop,timestamp 237390801 0> (DF)

22:21:32.505030 192.168.52.199.37914 > 204.79.197.203.80: S 3058318794:3058318794(0) win 16384 <mss
1460,nop,nop,sackOK,nop,wscale 6,nop,nop,timestamp 1892450804 0> (DF)
```

macOS

```
22:23:38.201671 IP 192.168.52.191.51747 > 52.109.120.23.443: Flags [S], seq 1792899115, win 65535, options [mss
1460,nop,wscale 6,nop,nop,TS val 181072898 ecr 0,sackOK,eol], length 0

22:23:39.363822 IP 192.168.52.191.51748 > 64.233.177.106.443: Flags [S], seq 2151761592, win 65535, options [mss
1460,nop,wscale 6,nop,nop,TS val 181074052 ecr 0,sackOK,eol], length 0

22:23:39.397858 IP 192.168.52.191.51749 > 64.233.177.106.443: Flags [S], seq 2082510982, win 65535, options [mss
1460,nop,wscale 6,nop,nop,TS val 181074084 ecr 0,sackOK,eol], length 0

22:23:39.804972 IP 192.168.52.191.51750 > 172.217.0.131.443: Flags [S], seq 1755714468, win 65535, options [mss
1460,nop,wscale 6,nop,nop,TS val 181074459 ecr 0,sackOK,eol], length 0
```

ldd

```
obsd-hack01$ ldd /usr/bin/opensync
```

```
/usr/bin/opensync:
```

Start	End	Type	Open	Ref	GrpRef	Name
000015dc40700000	000015dc40718000	exe	1	0	0	/usr/bin/opensync
000015de9991b000	000015de99b33000	rlib	0	1	0	/usr/lib/libcrypto.so.45.4
000015dedc727000	000015dedc756000	rlib	0	1	0	/usr/lib/libm.so.10.1
000015df30cb9000	000015df30dae000	rlib	0	1	0	/usr/lib/libc.so.95.0
000015deae284000	000015deae284000	ld.so	0	1	0	/usr/libexec/ld.so

```
obsd-hack01$ ldd /usr/bin/opensync
```

```
/usr/bin/opensync:
```

Start	End	Type	Open	Ref	GrpRef	Name
000000394c700000	000000394c718000	exe	1	0	0	/usr/bin/opensync
0000003b8c919000	0000003b8cb31000	rlib	0	1	0	/usr/lib/libcrypto.so.45.4
0000003bdd420000	0000003bdd44f000	rlib	0	1	0	/usr/lib/libm.so.10.1
0000003b727bb000	0000003b728b0000	rlib	0	1	0	/usr/lib/libc.so.95.0
0000003bb3396000	0000003bb3396000	ld.so	0	1	0	/usr/libexec/ld.so

Randomized Libraries

First boot

```
obsd-hack01$ sha256 /usr/lib/libc.so.95.0  
/usr/lib/libcrypto.so.45.4
```

```
SHA256 (/usr/lib/libc.so.95.0) =  
344809f88173408f69608893a34efac7c20dcbd1c4747d  
f9121c54825e9cb826
```

```
SHA256 (/usr/lib/libcrypto.so.45.4) =  
bcb29eb26b97899dbc98714a552950a34d11ba6bedf545  
6001fe290afd2791b0
```

```
obsd-hack01$ nm /usr/lib/libc.so.95.0 | fgrep  
strncpy
```

```
00079080 t _libc_strncpy
```

```
00079080 W strncpy
```

```
00000000 F strncpy.c
```

Second boot

```
obsd-hack01$ sha256 /usr/lib/libc.so.95.0  
/usr/lib/libcrypto.so.45.4
```

```
SHA256 (/usr/lib/libc.so.95.0) =  
797890c92725d1fe4e4b561d4b8a882657a5dbba481ea7  
5b44a6e664bcb079c1
```

```
SHA256 (/usr/lib/libcrypto.so.45.4) =  
e55b22519f286d69cc1c15b9d555914d56c032584a4e5f  
6e696c32179ec4131c
```

```
obsd-hack01$ nm /usr/lib/libc.so.95.0 | fgrep  
strncpy
```

```
0003d480 t _libc_strncpy
```

```
0003d480 W strncpy
```

```
00000000 F strncpy.c
```

KARL: Randomized unique kernel on boot

First boot

```
obsd-hack01# sha256 /bsd
SHA256 (/bsd) =
b044bb979cb974296b0cfcddb975eb7dc8fc92e1648381
f118b28de47b4607a1

obsd-hack01# nm /bsd | fgrep pf_test
ffffffff8193f5f0 T pf_test
ffffffff81937990 T pf_test_rule
ffffffff8193a5c0 T pf_test_state
ffffffff8193b600 T pf_test_state_icmp

obsd-hack01# reboot

...
```

Second boot

```
obsd-hack01# sha256 /bsd
SHA256 (/bsd) =
1e8027c41f5867026a45651faf90f4f6d74b9719869fe5
2229104e1769459e1c

obsd-hack01# nm /bsd | fgrep pf_test
ffffffff811910c0 T pf_test
ffffffff81189460 T pf_test_rule
ffffffff8118c090 T pf_test_state
ffffffff8118d0d0 T pf_test_state_icmp
```

Set up your OpenBSD
environment

Malloc options

Malloc option	What it stands for	What it does
C	Canaries	Add canaries at end of allocations to detect heap overflows
F	Freecheck	More extensive checks for: <ul style="list-style-type: none">• Double free• Use after free
G	Guard	Enable guard pages
J / j	More junking / Less junking	Increase/decrease junk level
R	Realloc	Always reallocate when realloc(3) is called
S		Enable all options for security auditing. Equivalent to “CFGJ”
U	Free unmap	Enable use after free protection for larger allocations
X	xmalloc	Make malloc() abort() instead of returning failure

How to use malloc options

- OpenBSD 6.5: `vm.malloc_conf` `sysctl`
 - `sysctl vm.malloc_conf=CFGJSUX`
 - Will enable system-wide malloc options, works with `chroot` and `unveil`
 - `MALLOC_OPTIONS` environment variable
 - `char *malloc_options` variable in program
-
- If you can't run them all the time, at least run them with your test suite
 - `MALLOC_OPTIONS=CFGJSUX ./run-tests.sh`

Make sure your program can survive W^X

- W^X – policy for the whole memory address space
 - A page is either writeable or executable, but not both
- Unfortunately some programs (e.g. Python, Ruby, Java, etc) need W|X
- On OpenBSD, such programs can run only if:
 - They are on a filesystem mounted with wxallowed (e.g. /usr/local on OpenBSD)
 - Their binaries are marked with wxneeded

```
/dev/sd0e on /usr/local type ffs (local, noatime, nodev, wxallowed)
```

- Make sure your program does not have W^X violations:
 - Run your program on a filesystem that is **NOT** mounted wxallowed
 - `sysctl kern.wxabort=1`

Dude, where's my core file?

- If your program changes to a different user (e.g. in a privilege dropping scenario) and it crashes, you won't find its core file
- To get its core file:

```
sysctl kern.nosuidcoredump=3  
mkdir /var/crash/programname
```

- Going forward, its core file will show up in `/var/crash/programname`.
- `/var/crash/programname` can be root-owned.

Run your code on a different platform

- OpenBSD runs on 13 platforms
 - alpha
 - amd64
 - arm64
 - armv7
 - hppa
 - i386
 - landisk
 - loongson
 - luna88k
 - macppc
 - octeon
 - sgi
 - sparc64
- Smoke out bugs involving:
 - Endianness
 - Strict alignment issues
 - 32-bit vs 64-bit
 - Signed char
 - Stack grows up/down/selectable
 - ILP32 vs LP64

Audit your code

General audit guidelines

- Use a consistent coding standard
 - Makes code easier to audit; less likely to miss bugs
 - No coding standard? Adopt upstream's coding standard if it makes sense
 - e.g. OpenBSD style(9) for C, use RuboCop for Ruby, use gofmt for Go, etc
- Eliminate classes of bugs
 - If you find a bug, look for the same bug throughout the tree and fix it everywhere
- Do not sacrifice security for the sake of backward compatibility!
- Make sure you are using functions properly
 - e.g. check their return values
 - OpenBSD man pages are a good place to start

malloc

Don't do this!

```
p = malloc(sizeof(struct foo));  
  
p->x = 123;  
  
/* do something with p->x */
```

- Return value of malloc is not checked!
- Pointer is not freed

Do this instead

```
p = malloc(sizeof(struct foo));  
if (p == NULL)  
    err(1, "malloc");  
  
p->x = 123;  
  
/* do something with p->x */  
  
free(p);
```

- Return value of malloc is checked!
- Pointer is freed

snprintf

Don't do this!

```
snprintf(buf, sizeof(buf), input);
```

Potential format string vulnerability

Or this!

```
snprintf(buf, sizeof(buf), "%s", input);
```

No return value check

Do this instead

```
int ret = snprintf(buf, sizeof(buf), "%s", input);  
if (ret == -1 || ret >= sizeof(buf))  
    errx(1, "snprintf error");
```

Checks for snprintf failure and overflow

Use secure API alternatives if available

- `strncpy/strncat` (instead of `strcpy/strcat`)
- `strtonum` (instead of `atoi` or `strtol`)
- `explicit_bzero`
- `reallocarray`
- `freezero`
- `arc4random` (instead of `rand`, `random`, `rand48`)

strncpy

Horrible! Ugh!

```
strcpy(buf, input);
```

Buffer overflow waiting to happen

Bad

```
strncpy(buf, input, sizeof(buf) - 1);  
buf[sizeof(buf) - 1] = '\0';
```

strncpy does not auto-add the NUL and is dangerously easy to misuse

Good

```
(void)strncpy(buf, input, sizeof(buf));
```

strncpy guarantees NUL-termination if there is room

Excellent

```
if (strncpy(buf, input, sizeof(buf)) >= sizeof(buf))  
    errx(1, "input is too long");
```

strncpy can detect truncation

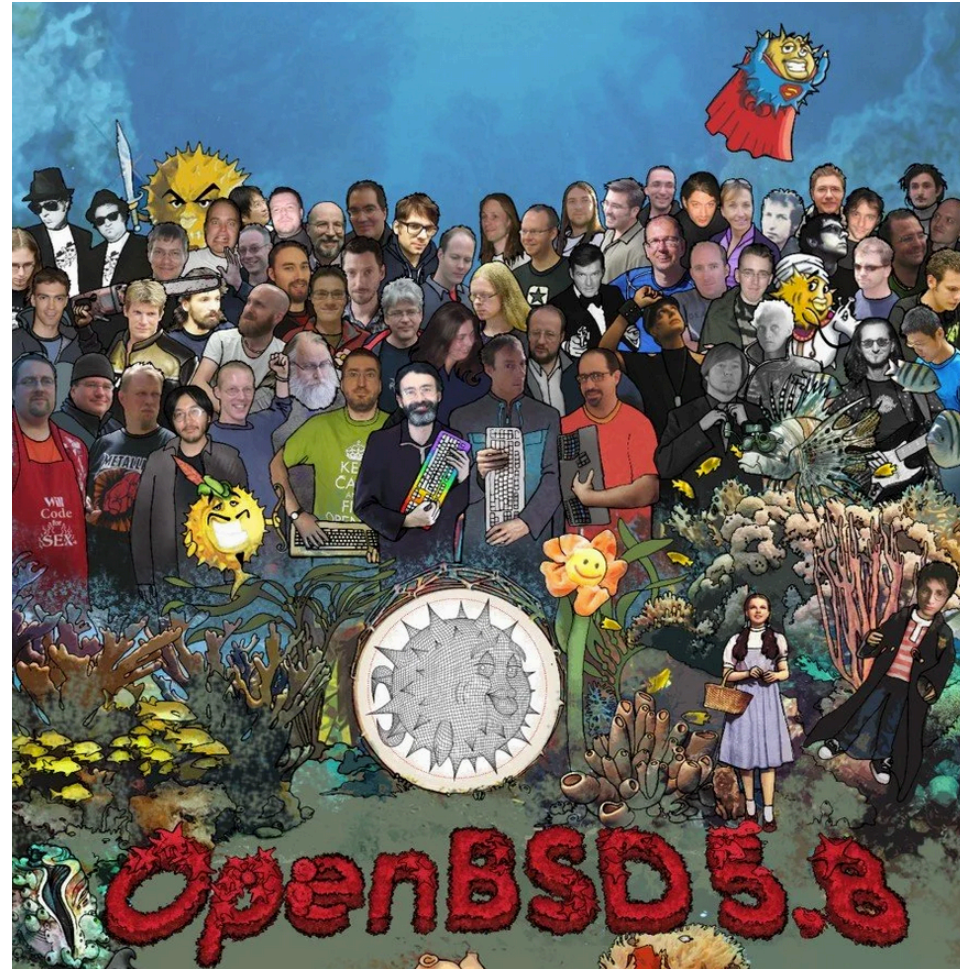
Working with TLS?

- Use libtls from LibreSSL
- Great for C programs that need to use common TLS methods
- NOT a Swiss army knife for everything TLS-related
- libtls tutorial
 - <https://github.com/bob-beck/libtls>



tedu your code

- Do not leave obsolete/legacy code lying around!
 - Breeding ground for bugs and technical debt
- Prune your code!
 - But be careful
- Bonus: You're reducing ROP gadgets too!
- Recommended reading:
 - <https://www.openbsd.org/papers/pruning.html>



Identify design issues

Identifying design issues

- `pledge(2)`
- `unveil(2)`
- Privilege dropping
- Privilege separation

pledge(2)

pledge(2)

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

- Restricts process to subsets of system calls
- Subsets are grouped into “promises”
- Some system calls, when allowed, will have additional restrictions
- Very easy to use
- Pledge will often expose design issues in your code
- Questions
 - How can I restrict my program to just use needed system calls?
 - Hoisting: Does it make sense to move more initialization code to the beginning so that the program can be pledged earlier?

pledge(2) promises (non-exhaustive)

Promise	System calls permitted	Notes/Conditions
stdio	System calls related to libc stdio	See pledge(2) man page for full list of system calls
rpath	chdir(2), getcwd(3), openat(2), fstatat(2), faccessat(2), readlinkat(2), lstat(2), chmod(2), fchmod(2), fchmodat(2), chflags(2), chflagsat(2), chown(2), fchown(2), fchownat(2), fstat(2), getfsstat(2)	Read-only effects on the filesystem
wpath	getcwd(3), openat(2), fstatat(2), faccessat(2), readlinkat(2), lstat(2), chmod(2), fchmod(2), fchmodat(2), chflags(2), chflagsat(2), chown(2), fchown(2), fchownat(2), fstat(2)	May cause write effects on filesystem
cpath	rename(2), renameat(2), link(2), linkat(2), symlink(2), symlinkat(2), unlink(2), unlinkat(2), mkdir(2), mkdirat(2), rmdir(2)	May create new files or directories
dpath	mkfifo(2), mknod(2)	Create special files
tmppath	lstat(2), chmod(2), chflags(2), chown(2), unlink(2), fstat(2)	Do operations in the /tmp directory, including create, read, or write

pledge(2) promises (non-exhaustive)

Promise	System calls permitted	Notes/Conditions
tmppath	lstat(2), chmod(2), chflags(2), chown(2), unlink(2), fstat(2)	Do operations in the /tmp directory, including create, read, or write
inet	socket(2), listen(2), bind(2), connect(2), accept4(2), accept(2), getpeername(2), getsockname(2), setsockopt(2), getsockopt(2)	Only in AF_INET and AF_INET6 domains; setsockopt(2) substantially reduced in functionality
unix	socket(2), listen(2), bind(2), connect(2), accept4(2), accept(2), getpeername(2), getsockname(2), setsockopt(2), getsockopt(2)	Only in AF_UNIX domain
dns	sendto(2), recvfrom(2), socket(2), connect(2)	Subsequent to a successful open(2) of /etc/resolv.conf, these system calls become able to allow DNS network transactions.

Unpledged process

```
graph TD; A([Unpledged process]) --> B[clock_getres(2), clock_gettime(2), close(2), closefrom(2), dup(2), dup2(2), dup3(2), fchdir(2), fcntl(2), fstat(2), fsync(2), ftruncate(2), getdents(2), getdtablecount(2), getegid(2), getentropy(2), geteuid(2), ...]; A --> C[chdir(2), getcwd(3), openat(2), fstatat(2), faccessat(2), readlinkat(2), lstat(2), chmod(2), fchmod(2), fchmodat(2), chflags(2), chflagsat(2), chown(2), fchown(2), fchownat(2), fstat(2), getfsstat(2)]; A --> D[socket(2), listen(2), bind(2), connect(2), accept4(2), accept(2), getpeername(2), getsockname(2), setsockopt(2), getsockopt(2)]; A --> E[Other system calls];
```

clock_getres(2),
clock_gettime(2), close(2),
closefrom(2), dup(2),
dup2(2), dup3(2),
fchdir(2),
fcntl(2), fstat(2), fsync(2),
ftruncate(2), getdents(2),
getdtablecount(2),
getegid(2), getentropy(2),
geteuid(2), ...

chdir(2), getcwd(3),
openat(2), fstatat(2),
faccessat(2), readlinkat(2),
lstat(2), chmod(2),
fchmod(2), fchmodat(2),
chflags(2), chflagsat(2),
chown(2), fchown(2),
fchownat(2), fstat(2),
getfsstat(2)

socket(2), listen(2), bind(2),
connect(2), accept4(2),
accept(2), getpeername(2),
getsockname(2),
setsockopt(2), getsockopt(2)

Other system
calls

```
if (pledge("stdio rpath", NULL) == -1)
    err(1, "pledge");
```

**Pledged
process**

stdio

rpath

X

X

clock_getres(2),
clock_gettime(2), close(2),
closefrom(2), dup(2),
dup2(2), dup3(2),
fchdir(2),
fcntl(2), fstat(2), fsync(2),
ftruncate(2), getdents(2),
getdtablecount(2),
getegid(2), getentropy(2),
geteuid(2), ...

chdir(2), getcwd(3),
openat(2), fstatat(2),
faccessat(2), readlinkat(2),
lstat(2), chmod(2),
fchmod(2), fchmodat(2),
chflags(2), chflagsat(2),
chown(2), fchown(2),
fchownat(2), fstat(2),
getfsstat(2)

socket(2), listen(2), bind(2),
connect(2), accept4(2),
accept(2), getpeername(2),
getsockname(2),
setsockopt(2), getsockopt(2)

Other system
calls

pledge(2) violation

```
obsd-hack01$ ./pledge-test
```

Abort trap (core dumped)

```
obsd-hack01$ dmesg
```

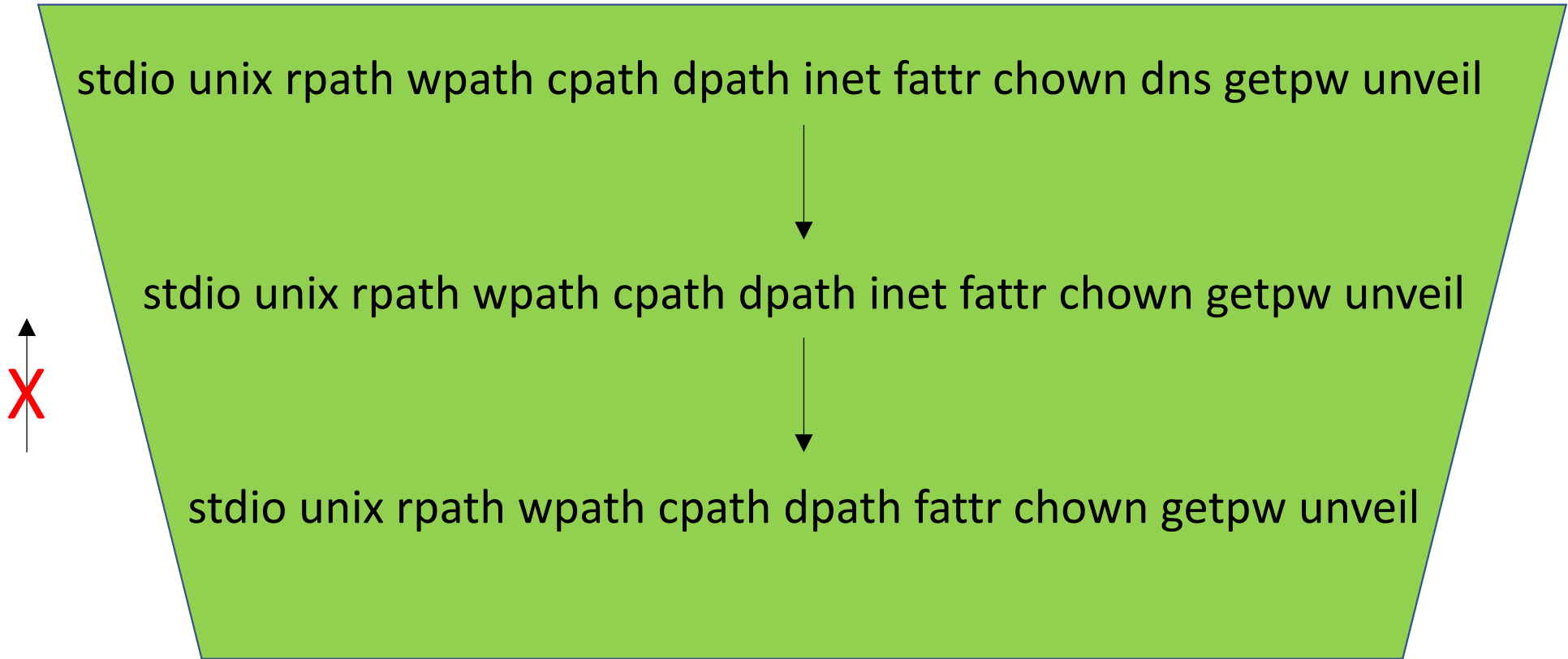
...

```
pledge-test[88667]: pledge "wpath", syscall 5
```

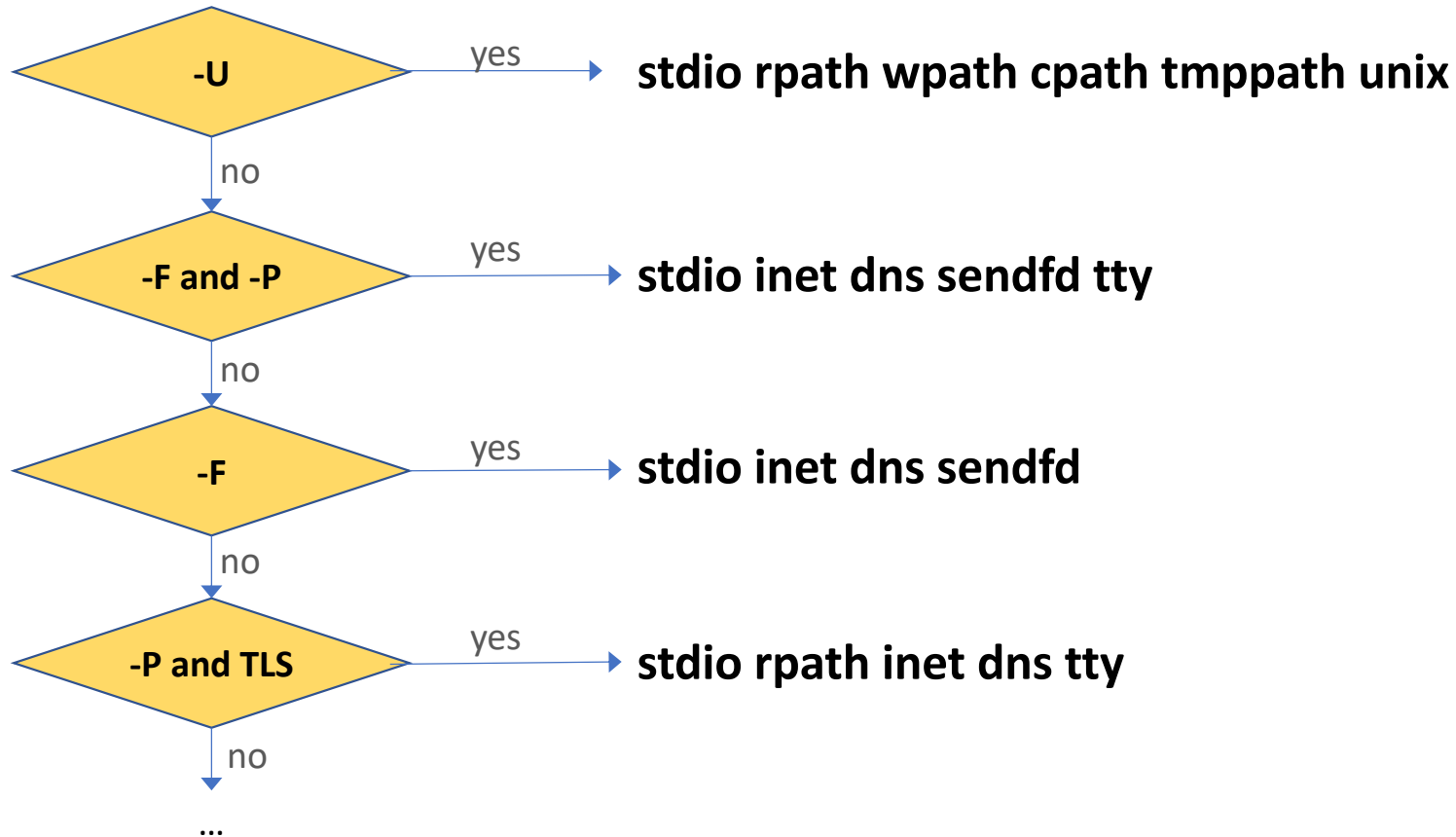
From /usr/src/sys/kern/syscalls.master:

```
5      STD      { int sys_open(const char *path, \
                  int flags, ... mode_t mode); }
```

Dropped promises cannot be regained



Another pledge(2) usage pattern e.g. netcat



If you have to do this...

```
if (pledge("stdio rpath wpath cpath dpath tmppath inet mcast fattr  
chown flock unix dns getpw sendfd recvfd tty proc exec ps id etc etc")  
== -1)  
    err(1, "pledge");
```

Your program could probably benefit from a redesign to use `privsep`

USER	PID	%CPU	%MEM	VSZ	RSS	TT	STAT	STARTED	TIME	COMMAND
root	1	0.0	0.0	472	420	??	Is	11:27AM	0:01.01	/sbin/init
root	58019	0.0	0.1	792	604	??	Isp	11:27AM	0:00.00	/sbin/slaacd
_slaacd	27945	0.0	0.1	796	704	??	Ip	11:27AM	0:00.00	slaacd: engine (slaacd)
_slaacd	43939	0.0	0.1	800	720	??	Ip	11:27AM	0:00.00	slaacd: frontend (slaacd)
root	94654	0.0	0.1	724	612	??	Is	11:27AM	0:00.02	dhclient: em0 [priv] (dhclient)
_dhcp	65591	0.0	0.1	852	724	??	Isp	11:27AM	0:00.06	dhclient: em0 (dhclient)
_syslogd	96793	0.0	0.1	1144	1508	??	Ip	11:27AM	0:00.12	/usr/sbin/syslogd
root	78417	0.0	0.2	584	2208	??	Isp	11:27AM	0:00.01	syslogd: [priv] (syslogd)
root	33569	0.0	0.1	848	636	??	Is	11:27AM	0:00.00	pflogd: [priv] (pflogd)
_pflogd	38905	0.0	0.1	892	588	??	Sp	11:27AM	0:02.41	pflogd: [running] -s 160 -i pflog0 -f /var/log/pflog (pflogd)
root	67045	0.0	0.2	836	1676	??	I<sp	11:27AM	0:00.07	/usr/sbin/ntpd
_ntp	36313	0.0	0.3	1252	2656	??	S<p	11:27AM	0:02.95	ntpd: ntp engine (ntpd)
_ntp	12397	0.0	0.2	748	2504	??	Ip	11:27AM	0:00.01	ntpd: dns engine (ntpd)
root	59212	0.0	0.1	1340	1432	??	Is	11:27AM	0:00.01	/usr/sbin/sshd
root	87362	0.0	0.2	1800	2116	??	Isp	11:27AM	0:00.00	/usr/sbin/smtpd
_smtpd	92263	0.0	0.4	1688	3812	??	Ip	11:27AM	0:00.01	smtpd: control (smtpd)
_smtpd	92202	0.0	0.3	1472	3584	??	Ip	11:27AM	0:00.01	smtpd: klondike (smtpd)
_smtpq	89904	0.0	0.4	1704	3796	??	Ip	11:27AM	0:00.01	smtpd: queue (smtpd)
_smtpd	83371	0.0	0.4	1556	3684	??	Ip	11:27AM	0:00.01	smtpd: lookup (smtpd)
_smtpd	26817	0.0	0.4	1620	3796	??	Ip	11:27AM	0:00.01	smtpd: pony express (smtpd)
_smtpd	42860	0.0	0.3	1476	3612	??	Ip	11:27AM	0:00.02	smtpd: scheduler (smtpd)
_sndio	58390	0.0	0.1	452	784	??	I<sp	11:27AM	0:00.00	/usr/bin/sndiod
_sndiop	80615	0.0	0.1	424	828	??	Isp	11:27AM	0:00.00	sndiod: helper (sndiod)
root	64792	0.0	0.1	696	1300	??	Isp	11:27AM	0:00.13	/usr/sbin/cron
root	69326	0.0	0.1	332	1280	C0	Is+p	11:27AM	0:00.00	/usr/libexec/getty std.9600 ttyC0
root	94253	0.0	0.1	336	1300	C1	Is+p	11:27AM	0:00.00	/usr/libexec/getty std.9600 ttyC1

p in STAT column indicates
a pledged process

pledge(2) bindings for other languages

- Node pledge
 - <https://github.com/qbit/node-pledge>
- Perl pledge
 - <https://github.com/afresh1/OpenBSD-Pledge>
- Ruby pledge
 - <https://github.com/jeremyevans/ruby-pledge>
- Rust pledge
 - <https://crates.io/crates/pledge>
- Others exist

unveil(2)

unveil

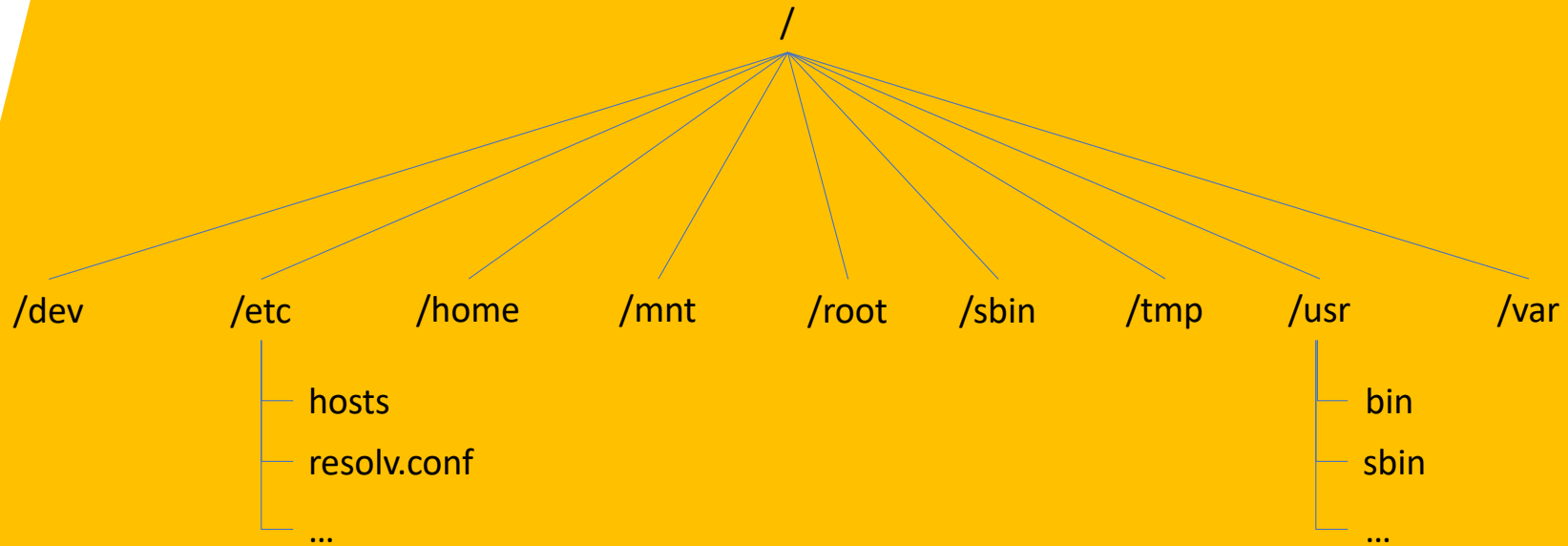
int

```
unveil(const char *path, const char *permissions);
```

- Removes visibility of entire filesystem except the specified path and permissions
- Permissions are r (read), w (write), x (execute), c (create)
- Relatively new: OpenBSD 6.4 (October 2018)
- Questions
 - Which files/directories does the program absolutely need to function?

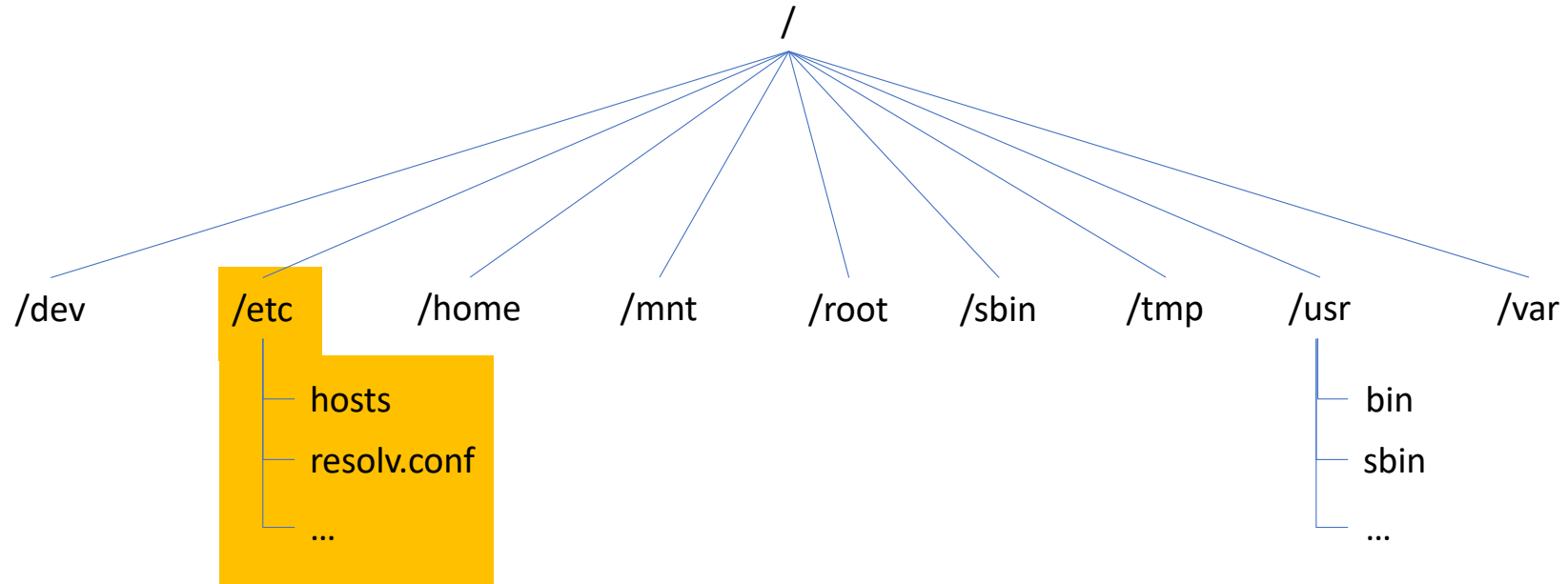
unveil(2) example

What your program sees when it starts



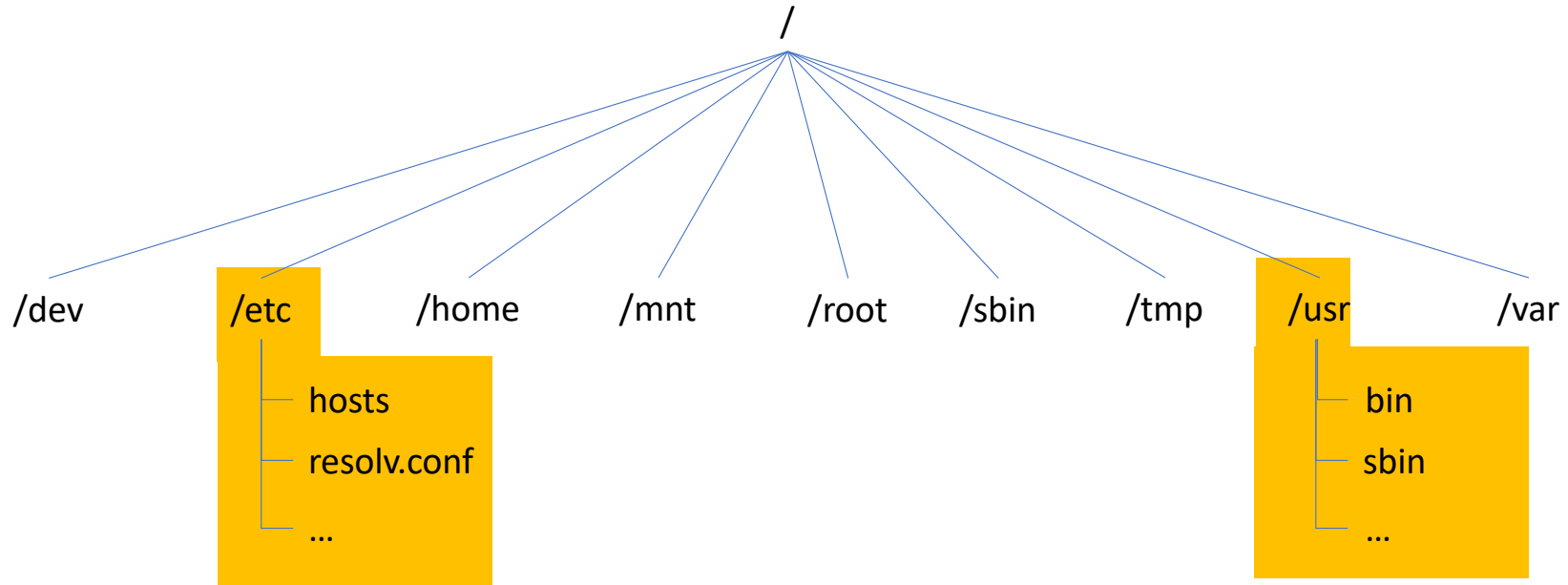
unveil(2) example

```
if (unveil("/etc", "r") == -1)  
    err(1, "unveil");
```



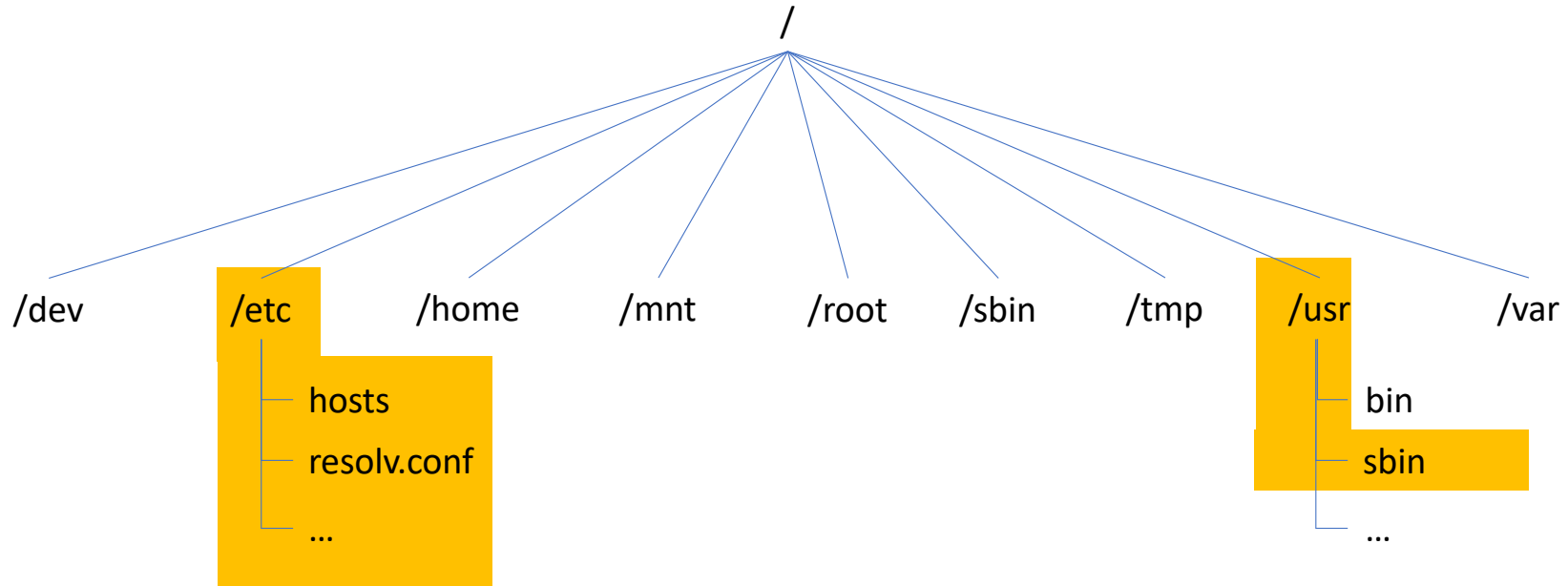
unveil(2) example

```
if (unveil("/usr", "r") == -1)  
    err(1, "unveil");
```



unveil(2) example

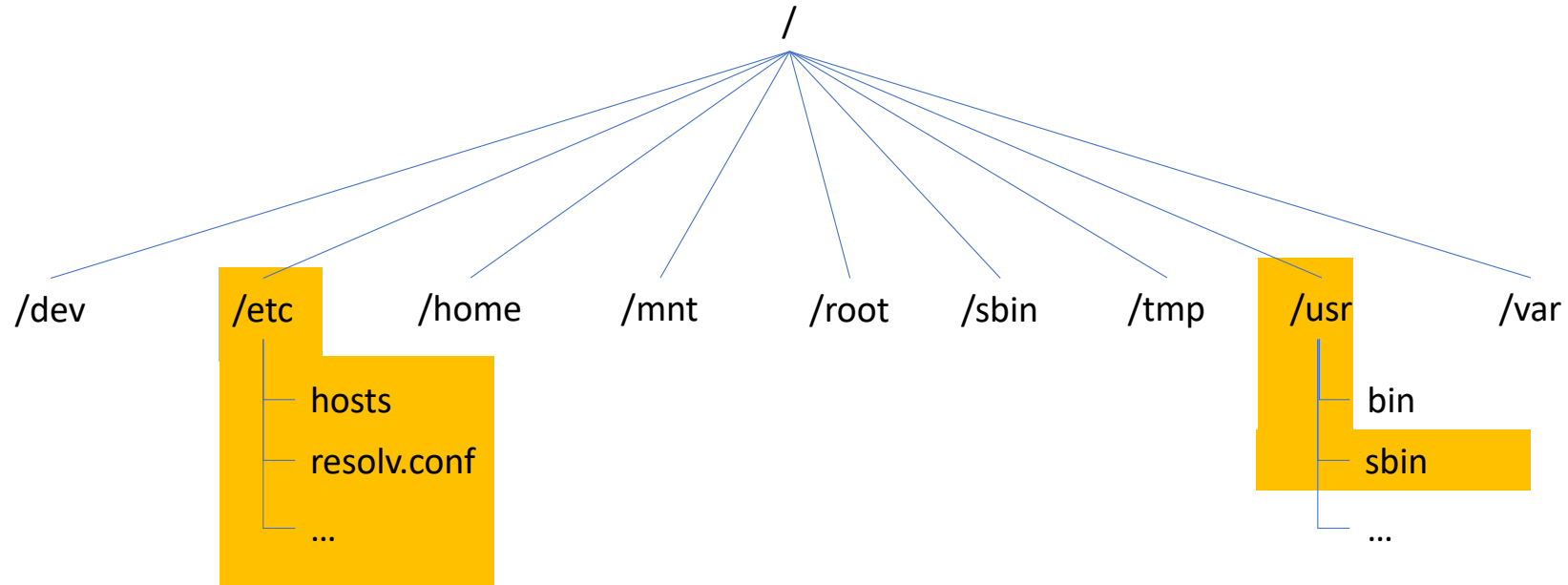
```
if (unveil("/usr/sbin", "r") == -1)  
    err(1, "unveil");
```



unveil(2) example

```
if (unveil(NULL, NULL) == -1)  
    err(1, "unveil");
```

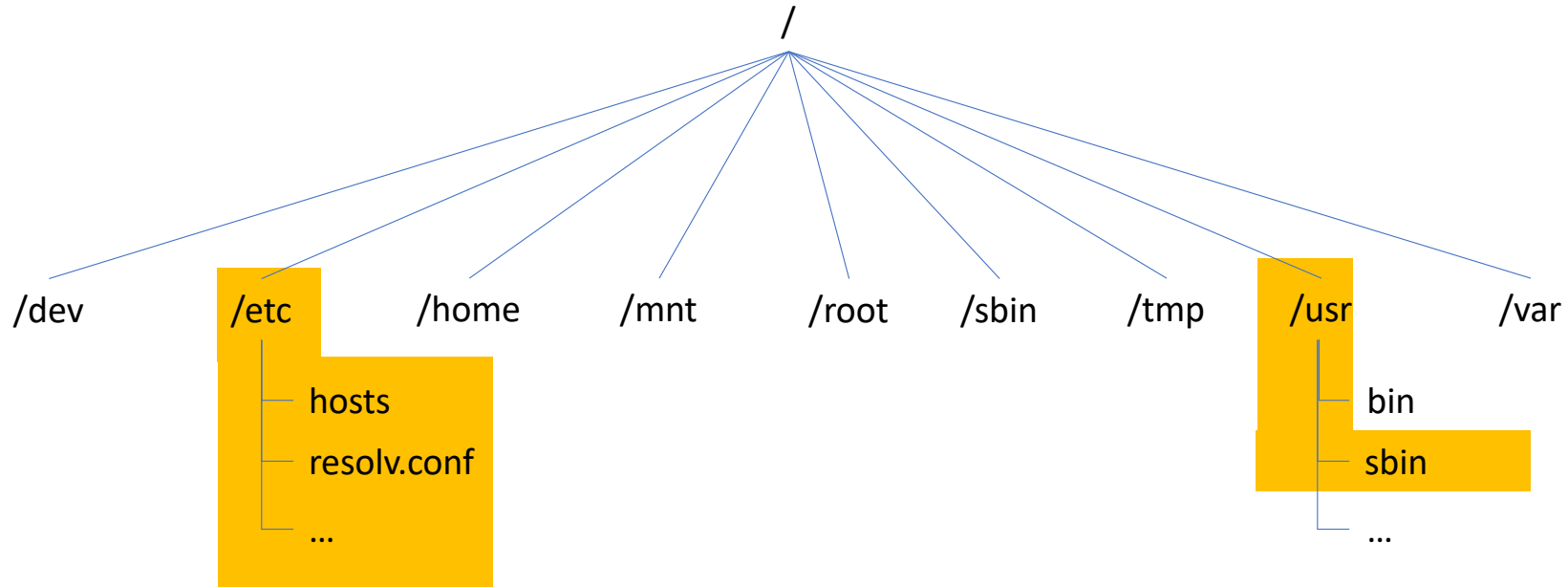
This disables further calls to unveil(2)



unveil(2) example

```
if (unveil("/var", "r") == -1)  
    err(1, "unveil");
```

unveil: Operation not permitted



unveil(2)

- Chromium on OpenBSD uses unveil
 - No access to your SSH keys!
- unveil(2) bindings
 - Ruby unveil
 - <https://github.com/jcs/ruby-unveil>
 - Rust unveil
 - <https://crates.rs/crates/unveil>

Privilege Dropping

Privilege dropping

- Principle of least privilege
- Questions
 - Does your code need to run as root?
 - Does it need to run as root all the time?
- Basics of privilege dropping
 - Create a dedicated non-root user for your program
 - Move code that require root to the beginning of the initialization routine
 - Open files, sockets, etc that require root
 - Chroot to an empty directory
 - Set the real, effective and saved user IDs to the dedicated non-root user

```

tzset();

pw = getpwnam("_foo");
if (pw == NULL)
    err(1, "getpwnam");

/* Set the process group of the current process */
if (setpgid(0, 0) == -1)
    err(1, "setpgid");

if (chroot(pw->pw_dir) == -1)
    err(1, "chroot");
if (chdir("/") == -1)
    err(1, "chdir");

if (setgroups(1, &pw->pw_gid) ||
    setresgid(pw->pw_gid, pw->pw_gid, pw->pw_gid) ||
    setresuid(pw->pw_uid, pw->pw_uid, pw->pw_uid))
    errx(1, "cannot drop privileges");

```

Privilege dropping example

tzset() grants you access to the local timezone after chroot. You can do this in scripting languages too, e.g. in Ruby, call Time.now

_foo is a user with /sbin/nologin as its shell, and an empty directory /var/foo as its home directory

```

$ finger _foo
Login: _foo                      Name: Foo
Directory: /var/empty           Shell:
/sbin/nologin
Never logged in.
No Mail.
No Plan.

```

Chrooting is not complete with just chroot() alone! chdir("/") after chroot() is required.

Privilege Separation




Privilege separation

- Principle of least privilege
- Questions
 - Does your program need to do something privileged way after initialization?
 - OpenBSD's ntpd root process: settimeofday()
 - Unprivileged processes do everything else
 - Does your program need to work with untrusted input?
 - OpenBSD's tcpdump: unprivileged process parses pcap files/input
 - Does your program need to work with something sensitive?
 - relayd and smtpd: separate process handles RSA private keys. Prevents HeartBleed-like exploits!

Privilege separation

- Basics of privilege separation
 - Create a dedicated non-root user for your program
 - Separate different responsibilities of your program into multiple processes
 - Use fork+exec to spawn each process
 - Restrict each process using privilege dropping, `pledge(2)`, `unveil(2)`, ...
 - Let the processes communicate via pipes




ntpd example

USER	PID	STAT	COMMAND	
root	365	I<sp	/usr/sbin/ntpd	 Calls settimeofday()
_ntp	75810	S<p	ntpd: ntp engine (ntpd)	 Speaks NTP to Internet
_ntp	60709	Ip	ntpd: dns engine (ntpd)	 Does DNS lookups

Privilege separation considerations

- Privilege separation + pledge + unveil
 - Unprivileged chrooted process could still access system calls it doesn't need
 - So restrict it with pledge(2)!
 - And restrict its filesystem with unveil(2)!
- Always use fork+exec to spawn the child processes
 - fork() alone will duplicate memory layout
 - fork+exec will randomize ASLR and stack cookies in the new process
 - Also protects against Blind ROP
 - switchd example
 - <https://github.com/openbsd/src/commit/9670b02bdf2c9449c661bab7e8693bdfb70d2e89>

ntpd with pledge promises

USER	PID	STAT	COMMAND	
root	365	I<sp	/usr/sbin/ntpd	 Calls settimeofday() stdio rpath inet settime proc exec id
_ntp	75810	S<p	ntpd: ntp engine (ntpd)	 Speaks NTP to Internet stdio inet
_ntp	60709	Ip	ntpd: dns engine (ntpd)	 Does DNS lookups stdio dns

OpenBSD programs using privsep (non-exhaustive list!)

- bgpd
- dhclient
- dhcpcd
- dvmrpd
- eigrpd
- file
- httpd
- iked
- ldapd
- ldpd
- lpd
- mountd
- npppd
- ntpd
- ospfd
- ospf6d
- pflogd
- pkg_add
- rad
- radiusd
- relayd
- slaacd
- script
- smtpd
- snmpd
- sshd
- switchd
- syslogd
- tcpdump
- tmux
- unwind
- vmd

Where to begin with privsep?

- Look at OpenBSD's ntpd
- Ken Westerback's (krw@) skeletal OpenBSD daemon:
 - <https://github.com/krwesterback/newd>

Recap

- Set up your OpenBSD environment
- Port your code to OpenBSD
- Audit it
- Remove legacy code
- Use `pledge(2)` and `unveil(2)` to identify design issues
- See if you can drop privileges
- See if it makes sense to use privilege separation

What next?

- Live the dream!
 - Write for OpenBSD, deploy on OpenBSD
- Write your code using OpenBSD's development practices
- Include OpenBSD as one of your program's target platforms
- Introduce privsep to other programs
- Port OpenBSD's exploit mitigation features to other systems
- Donate!
 - <http://www.openbsdoundation.org/>

Resources

- OpenBSD papers and slides
 - <https://www.openbsd.org/events.html>
- OpenBSD list of innovations
 - <https://www.openbsd.org/innovations.html>
- OpenBSD Journal
 - <https://www.undeadly.org/>
- Tweets on new OpenBSD features and exploit mitigation techniques
 - Bryan Steele [@canadianbryan](#)
- References for this talk
 - <https://lteo.net/carollinacon15>

Questions?

Lawrence Teo

lteo@openbsd.org

@lteo

Slides and references at:

<https://lteo.net/carolinacon15>