RBAC

Alistair Crooks
agc@netbsd.org
c0596823

Why do we do this?

- BSD Licensing?
- Belief that BSD combined kernel and userspace is correct distribution model?
- Because design is done right?
- Respect for BSD developers?

Security Models

- Mandatory Access Control
- Discretionary Access Control
- Role-Based Access Control

Effort to plug leaks

- All efforts put into stopping access
- root is superuser
- root can do anything

LEASE

Least privilege

- Only give as much privilege as is needed
- Why does ntpd need to run as root?
- Why does ping need to run as root?

RBAC

- No super-user
- Split privileges into roles
- Only use the least privilege role necessary

su & sudo

- We get least privilege through sudo
- Possibly
- More likely it’s superuser privileges on a per-user basis
- Not least privilege

Why GNU `su' does not support the `wheel' group

Sometimes a few of the users try to hold total power over all the rest. For example, in 1984, a few users at the MIT AI lab decided to seize power by changing the operator password on the Twenex system and keeping it secret from everyone else. (I was able to thwart this coup and give power back to the users by patching the kernel, but I wouldn't know how to do that in Unix.)

However, occasionally the rulers do tell someone. Under the usual `su' mechanism, once someone learns the root password who sympathizes with the ordinary users, he or she can tell the rest. The "wheel group" feature would make this impossible, and thus cement the power of the rulers.

I'm on the side of the masses, not that of the rulers. If you are used to supporting the bosses and sysadmins in whatever they do, you might find this idea strange at first.

-- Richard Stallman

Back to RBAC

LINKEY
KE2494

Abstraction and Indirection

A famous aphorism of Butler Lampson goes: *All problems in computer science can be solved by another level of indirection*; this is often deliberately mis-quoted with "abstraction" substituted for "indirection". Kevlin Henney's corollary to this is, "...except for the problem of too many layers of indirection."

Abstraction

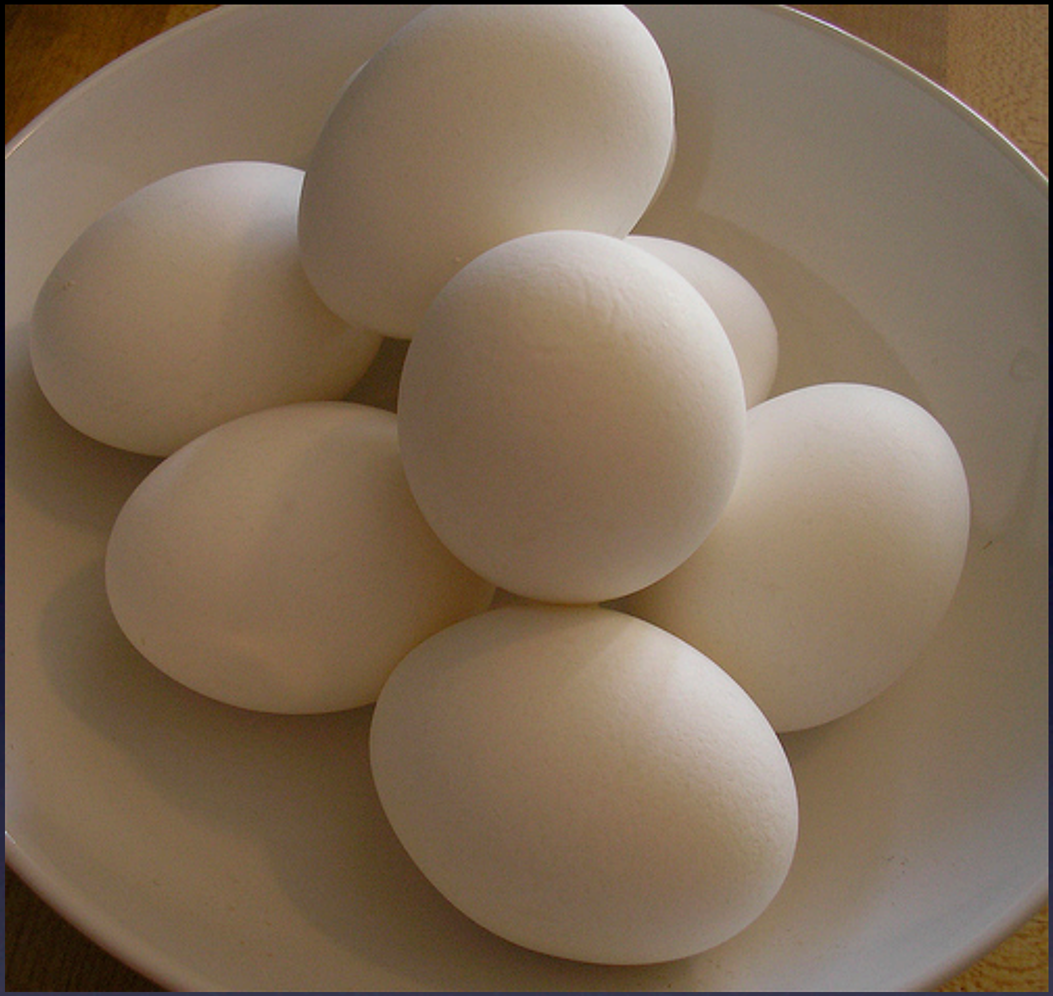
- Currently, we have a nanny state
- Root does what’s best for us
- Some people can get to play root
- and do anything they like

Abstraction

- We could split up the jobs into tasks
- Opportunity to do tasks they’re good at

Abstraction

- Give people what they need to do the task



Cornwall

Exploits

- ntpd, sshd, named
- chroot jails
- running with elevated privileges

Another way?

- Take away ability to do everything
- Do one thing, and do it well
- Not just users, programs that do things on our behalf - setuid root binaries

Edet

Roles

- If want to open a raw socket, don't need to be able to format the disks
- If want to set time, don't need to be able to trace processes
- We chroot-jail some processes - why not least privilege for all applications?

Identify Roles

- kauth does this for us
- 57 distinct roles in the form of kauth actions
- map kauth actions onto roles

kauth

- No more *issuser()*
- Instead calling sites call kauth functions
- kauth decides to allow, deny or defer
- group logically related choices together

Visual Differences

```
bsd44% ls -al /sbin/ping
-r-sr-xr-x 1 root wheel 68448 29 Nov 2007 /sbin/ping
bsd44%
```

```
rbac% ls -al /sbin/ping
-r-xr-sr-x 1 root net_open_sockraw_role_ 68448 29 Nov 2007 /sbin/ping
rbac%
```

Get raw socket - bsd44

```
bool isroot;

isroot = (kauth_cred_geteuid(cred) == 0);
...
case KAUTH_REQ_NETWORK_SOCKET_RAWSOCK:
        if (isroot)
                result = KAUTH_RESULT_ALLOW;
        break;
```

Get raw socket - rbac

```
case KAUTH_REQ_NETWORK_SOCKET_RAWSOCK:
    if (role_allows(cred, RBAC_SOCKET_RAWSOCK))
        result = KAUTH_RESULT_ALLOW;
    break;
```

role_allows()

```
static int
role_allows(kauth_cred_t cred, int role)
{
	int	ismember = 0;

	return kauth_cred_gid_has_role(
			cred, role, &ismember) == 0 && ismember;
}
```

kauth_cred_gid_has_role()

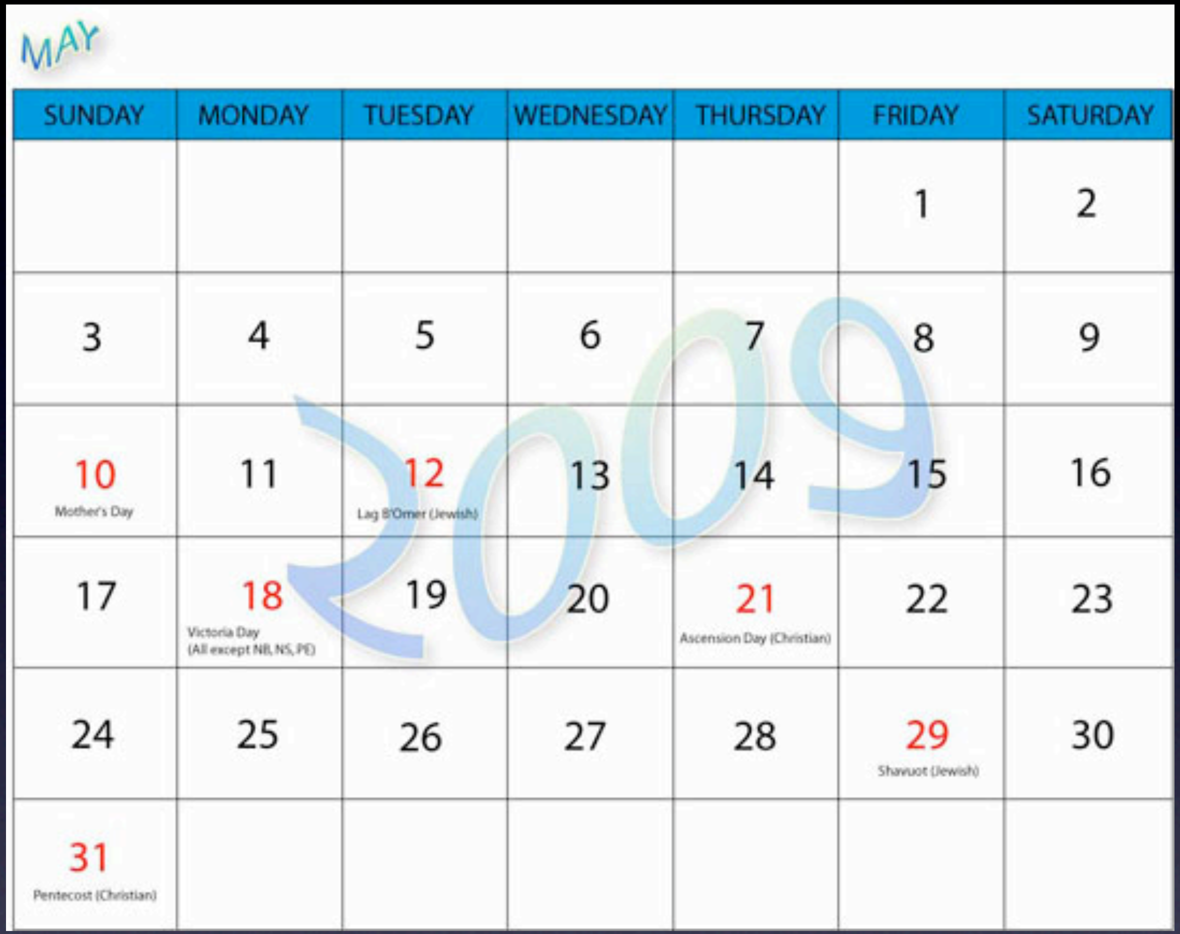
```
int kauth_cred_gid_has_role(kauth_cred_t cred,   gid_t roleneeded,   int *resultp)
{
        int     scope;
        int     role;
        int     i;

        *resultp = 0;

        scope = ROLE_SCOPE(roleneeded);
        role = ROLE_MASK(roleneeded);
        for (i = 0; i < cred->cr_ngroups; i++) {
                if (scope != 0 && ROLE_SCOPE(cred->cr_groups[i]) == scope) {
                        if (ROLE_MASK(cred->cr_groups[i]) & role) {
                                *resultp = 1;
                                break;
                        }
                }
        }
        return 0;
}
```

ROLE_SCOPE and ROLE_MASK

```
#define ROLE_SCOPE(r)   ((r) & 0xff000000)
#define ROLE_MASK(r)    ((r) & 0x00ffffff)
```

Socket Scope

```
/* specific socket roles */
RBAC_SOCKET_BIND_PRIVPORT     = 0x0c000001,
RBAC_SOCKET_OPEN_SOCKRAW      = 0x0c000002,
RBAC_SOCKET_CANSEE            = 0x0c000004,
RBAC_SOCKET_RAWSOCK           = 0x0c000008,
RBAC_SOCKET_ALL               = 0x0c00000f
```



MAY 2009

SUNDAY	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY
					1	2
3	4	5	6	7	8	9
10 Mother's Day	11	12 Lag B'Omer (Jewish)	13	14	15	16
17	18 Victoria Day (All except NB, NS, PE)	19	20	21 Ascension Day (Christian)	22	23
24	25	26	27	28	29 Shavuot (Jewish)	30
31 Pentecost (Christian)						

With RBAC, not quite so rooty

```
rbac# id
uid=0(root) gid=0(wheel) groups=0(wheel),2(kmem),3(sys),
4(tty),5(operator)
rbac# mknod -m 600 node c 0 0
mknod: node: Operation not permitted
rbac#
```

make node - bsd44

```
bool isroot;

isroot = (kauth_cred_geteuid(cred) == 0);
...
case KAUTH_REQ_NETWORK_SOCKET_RAWSOCK:
        if (isroot)
                result = KAUTH_RESULT_ALLOW;
        break;
```

make node - rbac

```
case KAUTH_REQ_NETWORK_SOCKET_RAWSOCK:
        if (role_allows(cred, RBAC_SYS_MKNOD)) {
                result = KAUTH_RESULT_ALLOW;
        }
        break;
```

RBAC mknod example

```
rbac# id
uid=0(root) gid=0(wheel) groups=0(wheel),2(kmem),3(sys),
4(tty),5(operator)
rbac# mknod -m 600 node c 0 0
mknod: node: Operation not permitted
rbac# su - agc
$ id
uid=1000(agc) gid=1000(agc) groups=1000(agc),
1342177296(sys_mknod_role_)
$ mknod -m 600 node c 0 0
$ ls -al node
crw------- 1 agc agc 0, 0 Feb 4 06:19 node
$ exit
rbac# exit
```

Bring Up

Oh what fun we had...

- mknod - worked first time

Finding the time

- KAUTH_REQ_SYSTEM_TIME_ADJTIME
- RBAC_SYS_TIME_ADJTIME
- sys_time_adjtime_role_ and adjtime(2)
- src/bin/date/date.c
- src/usr.sbin/rdate/rdate.c
- src/usr.sbin/timed/correct.c
- src/usr.sbin/timed/timed.c

Mountain climbing

- mountall does a “mount -a” which does an update mount on everything mounted
- su :mountroot -c 'mount /' in /etc/rc.d/root (instead of just mount /)
- add the su :mountroot before mount -a in mountall

Are we there yet?

- booting
- third-party software
- finding userland utilities calling sites

AMBO
OZ

Rules for RBAC

- setuid binaries become setgid binaries
- sudo - give user group membership

Problem

- I can’t put a single user into 57 groups!

Problem 2

- No one else uses this!

Problem 3

- I don’t feel safe giving a role to a user!

Problem 4

- It’s not the Unix way!

What have we learned?

- very useful research
- surprisingly effective
  - large-scale deployments
  - can’t be modified easily

Disadvantages

- tightly coupled kernel and userland
- fileassoc might be better tool
- huge amount of work in userland/packages

Advantages

- more than 50% complete
- can tie down some tasks to be non-root
- partial solution works very well
- useful if userland is “tied-down”

Further work

- look at fileassoc(9) for attaching roles to setuid and setgid binaries
- move rest of actions into roles

Thanks

Alistair Crooks
agc@netbsd.org
c0596823

Questions?

Alistair Crooks
agc@netbsd.org
c0596823

What's the score?