

# BSD HACKS

*100 Industrial-Strength  
Tips & Tools*



O'REILLY®

*Dru Lavigne*

HACK  
#57

## Tighten Security with Mandatory Access Control

Increase the security of your systems with MAC paranoia.

Ever feel like your Unix systems are leaking out extra unsolicited information? For example, even a regular user can find out who is logged into a system and what they're currently doing. It's also an easy matter to find out what processes are running on a system.

For the security-minded, this may be too much information in the hands of an attacker. Fortunately, thanks to the TrustedBSD project, there are more tools available in the admin's arsenal. One of them is the Mandatory Access Control (MAC) framework.



As of this writing, FreeBSD's MAC is still considered experimental for production systems. Thoroughly test your changes before implementing them on production servers.

### Preparing the System

Before you can implement Mandatory Access Control, your kernel must support it. Add the following line to your kernel configuration file:

```
options MAC
```

You can find full instructions for compiling a kernel in “Strip the Kernel” [Hack #54].

While your kernel is recompiling, take the time to read `man 4 mac`, which lists the available MAC modules. Some of the current modules support simple policies that can control an aspect of a system's behavior, whereas others provide more complex policies that can affect every aspect of system operation. This hack demonstrates simple policies designed to address a single problem.

### Seeing Other Users

One problem with open source Unix systems is that there are very few secrets. For example, any user can run `ps -aux` to see every running process or run `sockstat -4` or `netstat -an` to view all connections or open sockets on a system.

The `MAC_SEEOTHERUIDS` module addresses this. You can load this kernel module manually to experiment with its features:

```
# kldload mac_seeotheruids
Security policy loaded: TrustedBSD MAC/seeotheruids (mac_seeotheruids)
```

If you'd like this module to load at boot time, add this to `/boot/loader.conf`:

```
mac_seeotheruids_load="YES"
```

If you need to unload the module, simply type:

```
# kldunload mac_seeotheruids
Security policy unload: TrustedBSD MAC/seeotheruids (mac_seeotheruids)
```

When testing this module on your systems, compare the before and after results of these commands, run as both a regular user and the superuser:

- `ps -aux`
- `netstat -an`
- `sockstat -4`
- `w`

Your before results should show processes and sockets owned by other users, whereas the after results should show only those owned by the user. While the output from `w` will still show which users are on which terminals, it will not display what other users are currently doing.

By default, this module affects even the superuser. In order to change that, it's useful to know which `sysctl` MIBs control this module's behavior:

```
# sysctl -a | grep seeotheruids
security.mac.seeotheruids.enabled: 1
security.mac.seeotheruids.primarygroup_enabled: 0
security.mac.seeotheruids.specificgid_enabled: 0
security.mac.seeotheruids.specificgid: 0
```



`sysctl` is used to modify kernel behavior without having to recompile the kernel or reboot the system. The behaviors that can be modified are known as *MIBs*.

See how there are two MIBs dealing with `specificgid`? The enabled one is off, and the other one specifies the numeric group ID that would be exempt if it were on. So, if you do this:

```
# sysctl -w security.mac.seeotheruids.specificgid_enabled=1
security.mac.seeotheruids.specificgid_enabled: 0 -> 1
```

you will exempt group 0 from this policy. In FreeBSD, the `wheel` group has a GID of 0, so users in the `wheel` group will see all processes and sockets.

You can also set that `primarygroup_enabled` MIB to 1 to allow users who share the same group ID to see each other's processes and sockets.

Note that while you can change these MIBs from the command line, you will be able to see them only with the appropriate kernel module loaded.

## Quickly Disable All Interfaces

ifconfig allows you to enable and disable individual interfaces as required. For example, to stop traffic on *ed0*:

```
# ifconfig ed0 down
```

To bring the interface back up, simply repeat that command, replacing the word down with up.

However, ifconfig does not provide a convenient method for stopping or restarting traffic flow on all of a system's interfaces. That ability can be quite convenient for testing purposes or to quickly remove a system from a network that is under attack. The MAC\_IFOFF module is a better tool for this purpose. Let's load this module and see how it affects the system:

```
# kldload mac_ifoff
Security policy loaded: TrustedBSD MAC/ifoff (mac_ifoff)
# sysctl -a | grep ifoff
security.mac.ifoff.enabled: 1
security.mac.ifoff.lo_enabled: 1
security.mac.ifoff.other_enabled: 0
security.mac.ifoff.bpfrecv_enabled: 0
```

By default, this module disables all interfaces, except the loopback *lo* device. When it's safe to reenable those interfaces, you can either unload the module:

```
# kldunload mac_ifoff
Security policy unload: TrustedBSD MAC/ifoff (mac_ifoff)
```

or leave the module loaded and enable the interfaces:

```
# sysctl -w security.mac.ifoff.other_enabled=1
security.mac.ifoff.other_enabled: 0 -> 1
```

Perhaps you have a system whose interfaces you'd like to disable at bootup until you explicitly enable them. If that's the case, add this line to */boot/loader.conf*:

```
mac_ifoff_load="YES"
```

## See Also

- `man 4 mac`
- `man mac_seeotheruids`
- `man mac_ifoff`
- `man sysctl`
- The TrustedBSD project (<http://www.trustedbsd.org/>)
- The sysctl section of the FreeBSD Handbook ([http://www.freebsd.org/doc/en\\_US.ISO8859-1/books/handbook/configtuning-sysctl.html](http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/configtuning-sysctl.html))

- The MAC section of the FreeBSD Handbook ([http://www.freebsd.org/doc/en\\_US.ISO8859-1/books/handbook/mac.html](http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/mac.html))