Controlled credentials transitions without privileges: `mac_do(4)`, `mdo(1)` and `setcred(2)`

Olivier Certner

Kumacom SARL

Sponsored by The FreeBSD Foundation

EuroBSDCon 2025

Involvement With FreeBSD

Private

- Using FreeBSD since 2004
- Using it everywhere I can
- Maintaining small private changes (ports, userland, kernel)

Public

- Since ~20 years: Sporadic bug reports and mails on lists
- Since ~4 years: Gradual increase in involvement
  - Maintaining a few ports
  - Reporting bugs in base and submitting patches
- Since ~2 years: Working full time
  - Contractor for the FreeBSD Foundation since 2023/09
  - Committer since ~1 year and a half (`olce@`)
  - Presented at AsiaBSDCon 2024, EuroBSDCon 2024, BSDCan 2025

Other Work (1/2)

- Login classes
- Process visibility
- Zenbleed mitigation
- Vnode recycling
- Scheduler
  - Priorities
    - Single runqueue with 256 queues (committed)
    - POSIX.1b and rtprio(2) interfaces bug fixes and evolutions (stalled)
  - Hybrid Architectures (P/E cores, big.LITTLE) (WIP)
  - Other evolutions (TODO)

Other Work (2/2)

- Graphics/DRM support
- `ps(1)`
- SMBIOS
- `unionfs(4)`
  - Revamp long term proposal (2 years ago)
  - Review Jason Harmening's (`jah@`) fixes (mostly last year)
  - Revamp continuously deferred, but to start eventually

This Talk

This Talk

1. Traditional Credentials Management
2. Principles and Initial Implementation

This Talk

This Talk

This Talk

Outline

Process Credentials

Data

- Per-process, inherited
- User ID (UID), primary group ID (GID)
  - Real, effective and saved variants
- Supplementary groups

Uses

- Discretionary Access Control (DAC)
  - File access
  - Shared memory
  - Message queues
  - Semaphores
- Signals (`kill(2)`)
- Resource limits

Credentials API

- Modifies the calling process only
- Needs “Appropriate privileges”
- `setuid(2)`, `seteuid(2)`, `setreuid(2)`(, `setresuid(2)`)
- `setgid(2)`, `setegid(2)`, `setregid(2)`(, `setresgid(2)`)
- `setgroups(2)`

How to Change Credentials?

- Start with a privileged process
  - `root` from the start
  - “set-user-ID” executables with `root` as the owner

How to Change Credentials?

- Start with a privileged process
  - `root` from the start
  - "set-user-ID" executables with `root` as the owner
- Call the API
  - Set primary groups
  - Set supplementary groups
  - Possibly set other attributes requiring privileges
  - Change user IDs
  - Change effective user ID last (drops privileges)

What Do Programs Establishing Credentials?

Authentication Source of the request?

Authorization Access control

Spawn `fork(2)`

Setup

- Attributes requiring privileges
  - Resource limits
  - Scheduling settings
- Credentials
- Other attributes
  - Environment variables (`PATH`, `USER`, ...)
  - umask

Handover `exec(3)`

Example Programs

`su(1)` "Substitute Identity". Traditional, simple.

- Takes a login name
- Asks for a password
- Invokes a shell

Example Programs

`su(1)` "Substitute Identity". Traditional, simple.

- Takes a login name
- Asks for a password
- Invokes a shell

`sudo(8)` The swiss-army knife

- Policy plugins (`sudoers(5)`)
- Authenticates with caller's password
- Fine-grained control on commands
  - `exec(3)` interception (via `LD_PRELOAD`)
- Credentials caching
- Full events and I/O logging

Example Programs

`su(1)` "Substitute Identity". Traditional, simple.

- Takes a login name
- Asks for a password
- Invokes a shell

`sudo(8)` The swiss-army knife

- Policy plugins (`sudoers(5)`)
- Authenticates with caller's password
- Fine-grained control on commands
  - `exec(3)` interception (via `LD_PRELOAD`)
- Credentials caching
- Full events and I/O logging

`doas(1)` An intermediate take

- Simple transition rules
- Filters on the executable
- Full login possible

Security?

- “set-user-ID” executables
- `sudo(8)` is big
- Security audit/compliance
- What if they are compromised?

Outline

Goals

- Credentials transitions without “set-user-ID” executables
- Allow selected transitions only
- Support role-based scenarios
- No need to install a third-party package

mac_do(4)/mdo(1)

Introduced by Baptiste Daroussin (bapt@)

mac_do(4)

- Kernel module based on the MAC framework (mac(4))
- Allows some credentials-changing system calls to succeed
- According to rules configured by the administrator
- Only for mdo(1) processes

mdo(1)

- Companion userland program
- Decide on the calls and make them

mac_do(4)'s Rules

From/Match Part

- Does the rule apply?
- Works on the caller's credentials
- Can match a single user ID or group ID

```
uid=1001
```

To/Target Part

- Which new credentials are admissible?
- Originally: Check for a single target user ID (or *)

Example (before)

```
uid=1001:1002,gid=0:0
```

mdo(1) (before)

Operation

- getpwnam()
- initgroups() (= getgrouplist() + setgroups())
- setgid()
- setuid()

mdo(1) (before)

Operation

- getpwnam()
- initgroups() (= getgrouplist() + setgroups())
- setgid()
- setuid()

Limitations

- Cannot set custom groups
- setgroups(), setgid() require privilege
- mac_do(4) allows these calls without checks

Outline

Groups Use Cases

- Supplementary groups as tags
  - To allow access (via DAC)
  - To deny access (via MAC, see, e.g., `mac_bsdextended(4)`)
  - Audit purposes
- Part of user's identity
  - Access to specific resources assumed to be granted

Selectively grant/revoke membership

New Groups Functionality

- Augment rules to specify authorized or denied groups
- “Primary” groups vs. supplementary ones
- Different groups directives per user

Rule To/Target Part

- Now, multiple target clauses, typed:

`uid`	User ID
`gid`	"Primary" group ID
`+gid`	Supplementary group ID
`!gid`	Mandatory supplementary group ID
`-gid`	Forbidden supplementary group ID

- Parameter: Numerical ID or `.` (dot) for current value
- Combinations
  - `uid`, `gid`, `+gid`: Disjunction
  - `!gid`, `-gid`: Conjunction

```
uid=1002,gid=1002,+gid=1002,+gid=48
```

mdo(1) (before)

Operation

- getpwnam()
- initgroups() (= getgrouplist() + setgroups())
- setgid() ← Which rule to match?
- setuid()

mdo(1) (before)

Operation

- getpwnam()
- initgroups() (= getgrouplist() + setgroups())
- setgid() ← Which rule to match?
- setuid() ← Is this the end?

setcred(2)

Change desired credentials at once

```
int setcred(u_int flags, const struct setcred *wcred,
                         size_t size);
```

SETCREDF_UID	Set the effective user ID
SETCREDF_RUID	Set the real user ID
SETCREDF_SVUID	Set the saved user ID
SETCREDF_GID	Set the effective group ID
SETCREDF_RGID	Set the real group ID
SETCREDF_SVGID	Set the saved group ID
SETCREDF_SUPP_GROUPS	Set the supplementary group list
SETCREDF_MAC_LABEL	Set the MAC label

setcred(2)

Benefits

- mac_do(4) can now see the full transition
- Prevent observing transient states

setcred(2)

Benefits

- mac_do(4) can now see the full transition
- Prevent observing transient states

Alternate Approach: Transactions?

Stack desired credentials change and finally commit them.

- Keep using the traditional system calls
- Requires extensive kernel modifications
  - Changing MAC hooks in incompatible ways
- New/updated credentials-related calls expected to be rare

mdo(1)

Operation (before)

- getpwnam()
- initgroups() (= getgrouplist() + setgroups())
- setgid()
- setuid()

mdo(1)

Operation (after)

- getpwnam()
- getgrouplist()
- setcred()

Rules Examples

```
uid=10001>uid=10002
uid=10001>uid=10002,gid=10002
uid=10001>uid=10002,gid=10002,+gid=.
uid=10001>uid=10002,gid=10002,!gid=.
uid=10001>uid=10002,gid=10002,+gid=.,-gid=10001
uid=10001>uid=10002,gid=*,+gid=*
gid=10001>uid=0
gid=10001>gid=10002,!gid=.
```

See the `mac_do(4)` manual page.

Configuration

- Through `sysctl(8)` knob `security.mac.do.rules`
- Through jail parameter `mac.do.rules`
- `sysctl(8)` knob and jail parameters values stay consistent (DWIM)
- Can inherit rules from the parent jail with `mac.do`

Outline

mdo(1): Changing Credentials (before)

Capabilities

- Requires a target user, root implicit
- On a name, also sets groups from the databases
- Can be instructed to preserve groups

Command-line

`-u <user>` Change user, and groups on name

`-i` Keep current groups

`<command>` With `<arg>...`

mdo(1): Changing Credentials (soon) (1/2)

Capabilities

- Backwards compatible
- Can set/override a primary group
- Can set/override, add to or substract supplementary groups
- Three exclusive modes for groups
  - Start from current groups
  - Start from password/group databases' ones
  - Complete explicit specification
- No need to change users
- Real, effective and save variants
- Foot-shooting prevention

mdo(1): Changing Credentials (soon) (2/2)

Command-line

- `-k` Keep current user, exclusive with `-u`
- `-g` Set (`-u`)/override (`-k`) primary groups
- `-G` Set/override the whole supplementary groups set
- `-s` Amend the supplementary groups set (add/remove groups)

`--Xuid` Override a specific user (`X` among `e,r,sv`)

`--Xgid` Override a specific primary group (`X` among `e,r,sv`)

Foot-shooting Prevention

- On `-u` with ID, complete group specification required
- All overrides imply an explicit `-u` (else `-k`)

mdo(1): Changing Credentials: Examples

```
mdo -u alice id
mdo -u 1001 -g wheel -G staff,operator sh
mdo -u bob -s +wheel,+operator id
mdo -k --ruid 1002 --egid 1004 id
```

Outline

The MAC Framework

- Customized system security
- Modules implementing hooks
- Objects and subjects labeling
- See `mac(4)`

Authorizing a System Call

Usual Steps

- Specific early deny hook
  - `MAC_POLICY_CHECK()`, `MAC_POLICY_CHECK_NOSLEEP()`
- `priv_check()`/`priv_check_cred()`

`mac_priv_check()`	Early common deny
`prison_priv_check()`	Jail early deny (not a hook)
	Super-user grants (not a hook)
`mac_priv_grant()`	A single policy is enough

`setuid(2)` Example

- `mac_cred_check_setuid(oldcred, uid)`
- `priv_check_cred(oldcred, PRIV_CRED_SETUID)`

`mac_do(4)`'s Requirements for `setcred(2)`

- Needs the expected final credentials
- Built from current ones
- When called by `priv_check_cred()`

setcred(2)'s MAC hooks/calls

```
mac_cred_setcred_enter()
mac_cred_check_setcred()
priv_check_cred()
mac_cred_setcred_exit()
```

setcred(2)'s MAC hooks/calls

- mac_cred_setcred_enter()
- mac_cred_check_setcred()
- priv_check_cred()
- mac_cred_setcred_exit()
- mac_do_setcred_enter()
- mac_do_check_setcred()
- mac_do_priv_grant()
- mac_do_setcred_exit()

setcred(2)'s MAC hooks/calls

`mac_do_setcred_enter()`

`mac_do_check_setcred()`

`mac_do_priv_grant()`

`mac_do_setcred_exit()`

- Allocate thread-local memory, store current rules
- Fill it with the new credentials
- Check if rules allow the transition
- Deallocate memory

Object-Specific Data

The Facility

- Embed arbitrary data in existing structures
- Extensible without changing the ABI
  - Add a new `struct osd` field
- Data organized in slots
- Fast access
- `struct jail`, `struct thread`
- See `osd(9)`

Use in `mac_do(4)`

- Jails: Store rules
- Threads: Store new credentials during `setcred(2)`

security.mac.do.rules

- SYSCTL_PROC() with CTLFLAG_PRISON (see sysctl(9))
- Read: Climb jails up to one with rules
- Write: Set rules on the current jail

Jail Parameters

- `mac.do`: Jail system node
  - Values: `new`, `inherit`, `disable`
  - Like `ip4`, `ip6`, `host`, `vnet`, ...
  - `SYSCTL_JAIL_PARAM_SYS_NODE()`, `SYSCTL_JAIL_PARAM_SYS_SUBNODE()`
- `mac.do.rules`: Same content as `sysctl(8)` knob `security.mac.do.rules`

Epilogue

Wrap Up

- Fine-grained support for UNIX credentials transitions (`mac_do(4)`)
- Per-jail configuration, supports inheritance
- Well documented
- Production ready
- New rules available in 14.3
- New credentials change requests in 15.0

Future Work - `mac_do(4)`

In Progress

- Configurable userland companion program paths
  - Thin jails scenarios
- Support traditional credentials-changing system calls

Future Work - `mac_do(4)`

In Progress

- Configurable userland companion program paths
  - Thin jails scenarios
- Support traditional credentials-changing system calls

Wish List

- Logging of why transitions are refused
- Specific configuration file?

Future Work - `mdo(1)`

In Progress

- Print the final request sent to the kernel
- New mode: Produce example rules for the requested transition

Future Work - `mdo(1)`

In Progress

- Print the final request sent to the kernel
- New mode: Produce example rules for the requested transition

Wish List

- Prompt for password
- Grow other functionalities of login-like programs?
  - PAM
  - Login classes

Future Work - `setcred(2)`

Wish List

- Ability to set more process attributes?
  - Scheduling settings
  - Login name

Thanks!

Sponsored by The FreeBSD Foundation

Questions? Thoughts?

olce@