Application-Defined Decentralized Access Control

Yuanzhong Xu, Alan Dunn, Owen Hofmann*, Michael Lee, Syed Akbar Mehdi, Emmett Witchel

UT-Austin, Google*
Access control mechanism

- Simplicity
  - Easy to understand
  - Less prone to bugs

- Flexibility
  - Expressive
  - Support many use cases
UNIX/Linux - simplicity

- Linux/UNIX
  - User: UID
  - Group: GID
  - Admin: root user
- Simplicity
  - Easy to understand
  - Less prone to bugs
UNIX/Linux - more flexibility

- Linux/UNIX
  - User: UID
  - Group: GID
  - Admin: root user
  - Simplicity
    - Easy to understand
    - Less prone to bugs
- Need more flexibility
  - setuid binary
  - effective UID
  - FS UID
  - sticky bit
  - ...

- More flexibility
setuid binaries make things tricky

suEXEC of Apache server: using setuid binaries to run CGI/SSI with different UIDs

“If you aren't familiar with managing setuid root programs and the security issues they present, we highly recommend that you not consider using suEXEC.”

— http://httpd.apache.org/docs/2.2/suexec.html

• Need more flexibility
  • setuid binary
  • effective UID
  • FS UID
  • sticky bit
  • …
Access control in server applications

- A server application typically uses its own, hand-crafted program logic to enforce access control.

Access control is hard to get right…

Source of bugs: among OWASP top 10 application security risks

https://www.owasp.org/index.php/Top_10_2010-Main

Why OS access control CANNOT help?
Inflexible OS-level access control

- Numerical identifiers for principals in a flat namespace
  - 32-bit integer UID, GID (Linux)

- centralized management of principals
  - root/administrator privilege required to manage users/groups
  - /etc/passwd, /etc/group

Consequences

- different servers/apps CANNOT manage principals separately
- requires mapping between server users and OS UIDs

- regular user CANNOT define an ad hoc group (like a circle in Google+)
- creating a server user requires modifying system-wide sensitive files
OS mechanism: **inflexible, but robust**

Used beyond basic access control of OS users

- Privilege separation in SSHD
  - different components have different UIDs

- Android
  - each application has a unique UID

- Mac OS X Seatbelt application sandbox
User/Application-defined access control

- Privilege separation
- Application sandbox
- Access control in server applications
- Flexible group sharing

A unified OS-level mechanism to support all those scenarios?

- a balance between Simplicity and Flexibility
DCAC

• DeCentralized Access Control

• Conceptually similar to traditional UNIX discretionary access control (DAC)

• But generalized, more flexible

• A unified model: familiar, intuitive
  - can naturally represent users/groups

• Coexists with DAC
## DCAC mechanisms summary

<table>
<thead>
<tr>
<th>DCAC basic attribute-based model</th>
<th>Augment privilege/access</th>
<th>Restrict privilege/access</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add child attribute</td>
<td></td>
<td>Drop attribute</td>
</tr>
<tr>
<td>Attribute gateway</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mechanisms for coexisting with DAC</th>
<th>Augment privilege/access</th>
<th>Restrict privilege/access</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grant access when EITHER DCAC OR DAC passes</td>
<td></td>
<td>pmask, UID-bit</td>
</tr>
</tbody>
</table>
DCAC Attributes - principal identifiers

- Attributes — hierarchically named strings - components separated by “.”

- `.u.alice` is the parent attribute of `.u.alice.photo`
  A parent attribute represents a superset of privileges of its child attributes
DCAC Attributes - principal identifiers

• Attribute is a generic abstraction — can represent different types of principals
  • OS users/groups
  • Server users/groups
  • Applications
  • Application components
• Naming conventions for OS users and groups:
  • user .u.<username>
  • group .g.<groupname>
DCAC processes vs Linux processes

DCAC Process

Attribute set
- .u.alice
- .u.alice.photos
- .g.students

Linux Process

UID: 1000
GIDs: 4, 20, ...

Inherited across fork() and exec()
DCAC objects vs Linux objects

DCAC File/IPC object

ACL: 4 access modes

- **read** = .g.students \lor .u.alice.photos
- **write** = .u.alice.photos
- **execute** = \emptyset
- **modify** = .u.alice

Each access mode is a formula of attributes in disjunctive normal form (DNF), w/o negations

Linux File/IPC object

permission bits

- `rw-xr-xr-x`

UID: 1000
(owner: alice)

GID: 100
(group: student)
## DCAC ACL access modes

<table>
<thead>
<tr>
<th>permission to read/write/execute</th>
<th>DCAC</th>
<th>traditional DAC (discretionary access control)</th>
</tr>
</thead>
<tbody>
<tr>
<td>read, write, execute access modes</td>
<td></td>
<td>rwx permission bits -rwxr-xr-x</td>
</tr>
<tr>
<td>Admin privilege — change access</td>
<td>modify access mode</td>
<td>UID (owner)</td>
</tr>
</tbody>
</table>

- A matching process can change ACL of the file
- A process with the same UID can chmod
Any process can change its attribute set

- **Deprivilege** a process without root:
  - A process can **add a child** attribute of any existing attribute in its attribute set
  - A process can always **drop** any attribute

- **Decentralized in privilege**, compared to Linux: `setuid()` syscall restricted to root
Example: sandbox a PDF viewer

Alice’s shell process

fork

.pdf viewer

fork add drop exec

read = .u.alice.pdf
Example: support server-defined users

2 requests from Alice (A) and Bob (B) are handled by the dispatcher process.

- The dispatcher process `apps.server` forks worker processes.
- Worker process for Bob sends a process request for Bob to `apps.server.u.bob`.
- Worker process for Alice sends a process request for Alice to `apps.server.u.alice`.

This diagram illustrates how requests are handled by server-defined users.
Augmenting a process’ privilege

Linux:

setuid binaries:

• e.g. sudo

  sudo allows a user’s process to become root, if the user is in group “admin”

DCAC:

Attribute gateways:

• e.g., represent an ad hoc group:

  the gateway allows a group member’s process to add the group attribute
Ad hoc group:

- created/managed by regular users
  - Who are members of the group?
  - Who are admins of the group?
Ad hoc group as gateway

Gateway — ACL for attribute 
.u.alice.reading-group

ACL

read = .u.chris \lor .u.david

modify = .u.alice \lor .u.bob

gateway: a special file

Group:
“reading-group” defined by Alice

members

admins

A B C D
Ad hoc group as gateway

Gateway — ACL for attribute `.u.alice.reading-group`

ACL

\[
\begin{align*}
read &= .u.chris \lor .u.david \\
modify &= .u.alice \lor .u.bob
\end{align*}
\]

child attribute of `.u.alice`: under control of Alice

Group: “reading-group” defined by Alice

members

admins

gateway: a special file
Ad hoc group as gateway

Gateway — ACL for attribute .u.alice.reading-group

ACL

read = .u.chris ∨ .u.david

modify = .u.alice ∨ .u.bob

Group: “reading-group” defined by Alice

members

admins

Regular members as the read access mode

gateway: a special file
Ad hoc group as gateway

Gateway — ACL for attribute .u.alice.reading-group

ACL

read = .u.chris V .u.david

modify = .u.alice V .u.bob

Group: “reading-group” defined by Alice

members

admins

Admins as the modify access mode:

Admin Privilege can modify this gateway, or create new gateways for .u.alice.reading-group

gateway: a special file
Ad hoc groups in a server application
Decentralized attribute gateways

• DCAC doesn’t enforce the location of gateways

• Specific OS distributions/applications should develop conventions
Decentralized attribute gateways

- Different applications manage their own gateways separately

Application 1

Application 2

Application 3

gateways
gateways
gateways
Coexisting with DAC in Linux

DAC: traditional discretionary access control

DCAC Process

Attribute set

.u.alice
.u.alice.photos
.g.students

Linux Process in DAC

UID: 1000 (alice)
GIDs: 4, 20, …
Coexisting with DAC in Linux

DAC: traditional discretionary access control

How to restrict DAC?

isolate processes with the same UID

Grant access when either DCAC or DAC passes

— A valid Linux disk image is a valid DCAC disk image (enables incremental deployment)

More permissive than DAC only

DCAC Process

Attribute set

.u.alice
.u.alice.photos
.g.students

UID: 1000 (alice)
GIDs: 4, 20, ...

How to restrict DAC?

isolate processes with the same UID

DAC: traditional discretionary access control

DCAC Process

Attribute set

.u.alice
.u.alice.photos
.g.students

UID: 1000 (alice)
GIDs: 4, 20, ...

More permissive than DAC only

— A valid Linux disk image is a valid DCAC disk image (enables incremental deployment)
How to restrict DAC?

DAC: traditional discretionary access control

DCAC Process

Attribute set

.u.alice  .u.alice.photos  .g.students

UID: 1000 (alice)
GIDs: 4, 20, …

pmask = 0555
UID-bit = 0

UID-bit
If UID-bit=0:
- give up the UID-based ambient authority
  e.g. DAC can’t allow chmod

A process can ONLY clear bits in pmask and UID-bit
  — to deprivilege itself

permission bits ANDed with pmask
e.g. pmask=0555: DAC can’t grant write permission

uid-bit

Bootstrap DCAC

- Attributes can only be added based on the current attribute set
- Who sets up the initial attribute?
  - Allow a root process to add any attribute
- Modify login/sshd/lightDM to set up attribute set of a user’s process with:
  - .u.<username>, with admin privilege
  - .g.<groupname>, without admin privilege
Represent ACLs on objects

- ACLs for **persistent files** are stored in **extended attributes** (xattr)

- ACLs are also **cached in memory**
  - support in-memory files, IPC objects
  - improve performance
  - can be invalidated by NFS according to time stamps, or hashes
Applications

A single model that supports these scenarios:

• A wrapper program that sets up a sandbox, for unmodified applications.

• DokuWiki [246 lines code change]
  - use DCAC to enforce access control
  - support ad hoc groups

• NFS [326 lines code change]
  - DCAC can operate on multiple machines
  - No centralized attribute server

• SSHD [81 lines code change]
  - Allow a regular OS user to define his/her sub-users, who can log in with a subset of the OS user's privilege.
Performance

• File system micro-benchmarks (Reimplemented Andrew Benchmarks, small file):

  DCAC only adds overhead on open, create, delete, etc.

• ext4:
  32B ACL:  under 4% slowdown
  - in-inode xattr
  256B ACL: under 9% slowdown
  - extra disk block for xattr

• NFSv3: under 5% slowdown
  ACL size has small impact on performance
  Extra round-trips (for fetching ACLs)
  — but not often, cached for most of the time
Performance

- Macro-benchmarks
  - Kernel compile: under 2% slowdown
    - both ext4 and NFSv3
  - DokuWiki: 0% slowdown
    - playing back 6,430 revisions of 765 pages to the DokuWiki website
Conclusion

- DCAC generalizes OS access control to support user/application-defined scenarios
- DCAC avoids the requirement of root privilege in many use cases
- DCAC does not require centralized attribute management
- DCAC coexists with DAC (discretionary access control)

Code available on GitHub: https://github.com/ut-osu/acac