Last Time - The Guard Model

Subject → Request → Guard → Objects

Is subject allowed to access resources?
Consider a GitHub-like Ecosystem

- Continuous Integration
- Git pages
- PR bot
- Autograder
- WWW
- Central code DB
- Apps access DB resources to provide extra services
- Application access must be restricted:
  - E.g. don’t make private repos public

Git repositories + code, user profiles, organizations
Let's Start with User Permissions

Associate a list of (user, permissions) with each resource

Repositories

cos316/assignment4-alevy.git

[(alevy, [PUSH, PULL]), (wlloyd, [PUSH, PULL]), (will, [PULL])]
### Implementing ACLs: Inline with Object

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>language</th>
<th>acl</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>cos316/assignment4-aalevy</td>
<td>Golang</td>
<td>“[(alevy, [PUSH,PULL]), (wlloyd, [PUSH,PULL]), ...]”</td>
</tr>
<tr>
<td>2</td>
<td>tock/tock</td>
<td>Rust</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
## Implementing ACLs: Normalize

### Repository Table

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>language</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>cos316/assignment4-aalevy</td>
<td>Golang</td>
</tr>
<tr>
<td>2</td>
<td>tock/tock</td>
<td>Rust</td>
</tr>
</tbody>
</table>

### ACL Table

<table>
<thead>
<tr>
<th>repo_id</th>
<th>user</th>
<th>permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>aalevy</td>
<td>push</td>
</tr>
<tr>
<td>1</td>
<td>kap</td>
<td>push</td>
</tr>
<tr>
<td>1</td>
<td>kap</td>
<td>pull</td>
</tr>
<tr>
<td>1</td>
<td>aalevy</td>
<td>pull</td>
</tr>
<tr>
<td>1</td>
<td>will</td>
<td>pull</td>
</tr>
<tr>
<td>2</td>
<td>aalevy</td>
<td>push</td>
</tr>
</tbody>
</table>

### Query

```sql
select (acls.user, acls.permission)
from repositories, acls
where repositories.name = 'cos316/assignment4-aalevy'
    and acls.repo_id = repositories.id;
```
ACLs in Action

```
select count(*) > 0
from repositories, acls
where
  repositories.name = 'cos316/assignment4-aalevy'
  and acls.repo_id = repositories.id
  and acls.user = 'aalevy'
  and acls.permission = 'push';
```

False?
Extending ACLs to Apps: a-la UNIX

- Applications act *on behalf of users*
- When an application makes a request, it uses a particular user’s credentials
  - Either one user per application
  - Or different users for different requests
- Works great for:
  - Alternative UIs, e.g. the `git` client vs. the GitHub Web UI both act on behalf of users
- Why might this be suboptimal?
Extending ACLs to Apps: Special Principles

- Create a unique principle for each app
  - E.g., the “autograder” principle
  - Acts just like a regular user

- When applications make requests, they use their own, unique, credentials

- Add application principals to resource ACLs as desired

- Works when
  - Applications need to operate with more than one user’s access
    - E.g. the autograder needs to access private repositories owned by different students
  - and less than any one user’s access
    - E.g. the autograder shouldn’t be able to access non COS316 repositories
Access Control Lists

Advantages

- Simple to implement
- Simple to administer
- Easy to revoke access

Drawbacks

- Tradeoff granularity for simplicity
  - More granular permissions require more complex rules in the guard
- Doesn’t scale well
  - E.g. need up to Users X Repos X Access Right entries in ACL table
- Centralized access control
  - Needs server’s cooperation to delegate access
An Alternative - Capabilities

“[A] token, ticket, or key that gives the possessor permission to access an entity or object in a computer system.” - *Capability-Based Computer Systems*

- **Self-describing**
  - Contains both object name and permitted operations
- **Globally meaningful**
  - Object and operation names are not subject-specific
- **Transferrable**
  - A subject can pass a capability to another (e.g. a sub-process, via IPC, a third-party app, etc)
  - Ideally can delegate subset of capabilities
- **Unforgeable**
  - Subjects cannot create capabilities with arbitrary permissions
File Descriptors as Proto-Capabilities

- **Unforgeable ✓**
  - Process-level fd is just an index in a kernel structure

- **Self-describing ✓**
  - Kernel fd contains reference to inode + permissions

- **Globally meaningful ✗**
  - Fds are process-specific

- **Transferrable ✓/ ✗**
  - Via IPC sendmsg/recvmsg
Consider a GitHub-like Ecosystem

- Continuous Integration
- Git pages
- PR bot
- Autograder

Central code DB
- Apps access DB resources to provide extra services
- Application access must be restricted:
  - E.g. don’t make private repos public
User Permissions using Capabilities

Hand out communicable, unforgeable tokens encoding:

- Object
- Access right

Users store capabilities, not the database

E.g.

“push(cos316/assignment4-aalevy)”

“pull(cos316/assignment4-aalevy)”
Implementing Capabilities with HMAC

HMAC - a keyed-hash function: \texttt{hmac(secret\_key, data)} hash of data

\begin{verbatim}
fn gen_capability(op, repo) {
    hmac(db_secret, fmt.Sprintf("%s(%s)", op, repo))
}

fn verify_capability(cap, op, repo) {
    cap == hmac(db_secret, fmt.Sprintf("%s(%s)", op, repo))
}
\end{verbatim}
Capabilities in Action

Doesn’t matter who

verify_capability(Cap, “push”, “cos316/assignment4-aalevy”)

Push(cos316/assignment4-aalevy, Cap)

Guard

cos316/assignment4-aalevy

Error!

False?
Extending Capabilities to Applications

- Users can simply give applications a subset of their capabilities
Extending Capabilities to Applications

```
Guard
```

```
verify_capability(Cap, "push",
    "cos316/assignment4-aalevy")
```

```
Push(cos316/assignment4-aalevy, Cap)
```

Error!

False?
Capabilities

Advantages

● Decentralized access control
  ○ Anyone can “pass” anyone a capability
● Scales well
● Granular permissions are simple to check

Drawbacks

● How do you revoke a capability?
● Moves complexity to users/clients
  ○ Users have to manage their capabilities now
Capabilities In The Wild

- Operating Systems
  - History of industry and research operating systems
  - seL4
  - FreeBSD’s Capsicum
  - Fuschia OS

- Web
  - S3 Signed URLs
    - URL to private resources, contain signature, expiration, permitted HTTP methods, etc
  - CDN-hosted images/videos (FB, Instagram, YouTube)
    - Browsing via Web page/app is protected by login+cookie, but media typically fetched unauthenticated
Next time...

We still have a problem!

The autograder is allowed to:

- read all cos316/ repositories
- comment on all cos316/ repositories

Can code from a private repository end up in a comment on a public repository?