Securing Access to Resources

COS 316: Principles of Computer System Design

Amit Levy & Ravi Netravali

Why might we want to control access to resources?

The Complete Guide to Facebook Privacy

Despite repeated privacy lapses, Facebook offers a fairly robust set of tools to control who knows what about you.

Facebook has never been particularly good at prioritizing your privacy. Your data powers its business, after all. But recent revelations that a firm called Cambridge Analytica harvested the personal information of 50 million unwitting Facebook users in 2015 has created new sense of urgency for those hoping for some modicum of control over their online life. If you ever needed a wake-up call, this is it.

The good news: Despite the repeated, public privacy lapses, Facebook does offer a fairly robust set of tools to control who knows what about you—both on the platform and around the web. The bad news: Facebook doesn't always make those settings easy to find, and they may not all offer the level of protection you want.

Fear not! Below, we'll walk you through the steps you need to take to keep advertisers, third-party apps, strangers, and Facebook itself at bay. And if after all that you still feel overly exposed? We'll show you how

—

Portland, Maine recognition ban

It's the latest city in the US to ban

Mariella Moon, @mariella_moon
7h ago

AndreyPopov via Getty Images

Election hoax spreading through text messages in Michigan

The text claims a 'typographical error' is switching peoples' votes

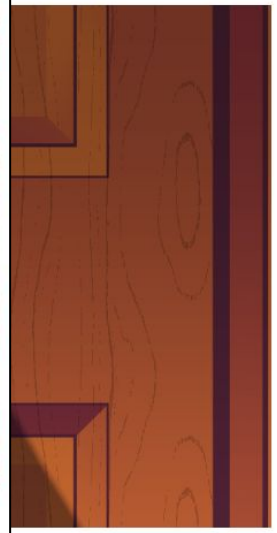
By Zoe Schiffer | @ZoeSchiffer | Nov 3, 2020, 2:16pm EST

SHARE

Illustrator by Alex Castro / The Verge

A text message campaign is targeting people in Michigan with misinformation about "ballot

on Ring

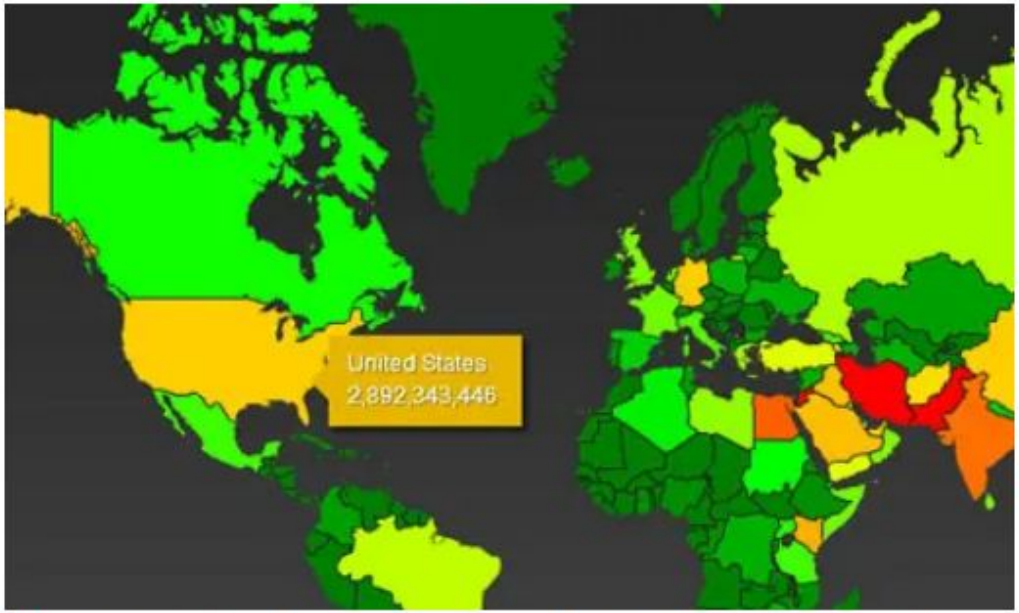


Why might we *not* want to control access to resources?

Boundless Informant: the NSA's secret tool to track global surveillance data

Revealed: The NSA's powerful tool for cataloguing global surveillance data – including figures on US collection

- **Boundless Informant: mission outlined in four slides**
- **Read the NSA's frequently asked questions document**



▲ The color scheme ranges from green (least subjected to surveillance) through yellow and orange to red (most surveillance). Note the '2007' date in the image relates to the document from which the interactive map derives its top secret classification, not to the map itself.

Designing secure systems is a subtle craft that must be informed by real-world, human, considerations.

A (slightly) formal model

- Objects: the things being accessed
  - A file, database table, network socket, satellite imagery of “nuclear facilities,” missile launcher...
- Subjects: an entity that requests access to an object
  - A process, network endpoint, etc…
  - **Principal**: some unique a account or role, such as a user
- Authentication: a proof that a subject *speaks for* some principal
  - E.g. logging in with a username & password
- Authorization: the particular rules that govern subjects’ access to objects
- Secrecy: who might learn the contents of an object
- Integrity: who may have influenced the contents of an object

Ad-hoc access control

- Access policy enforcement is scattered throughout system

```
fn (profile *Profile) viewProfile(user) (HTML) {
  if profile.public ||
     profile.friends.contains(user) {
    return profile.HTML
  } else {
    return HTML.Forbidden
  }
}
```

```
fn (profile *Profile) viewFullName(user) (HTML) {
  if profile.public || user.handle ==
                              “NSA_Backdoor” {
    return profile.FullName.HTML
  } else {
    return HTML.Forbidden
  }
}
```

- Very common in applications with lots of users. Why?

Ad-hoc access control

- Application-specific access rules
- Data for rules stored separately from data objects
  - Really a problem of granularity

Profile Table

id	full_name	profile_pic	handle	bio
1	Amit Levy	/i/1f3.png	alevy	Motivational speaker, futurist...
2	Wyatt Lloyd	/i/a60.png	wlloyd	Enjoys long function names...

Friends Table

follower	followee
1	2
2	1
1	4
1	5
...	...

Problems Ad-hoc access control

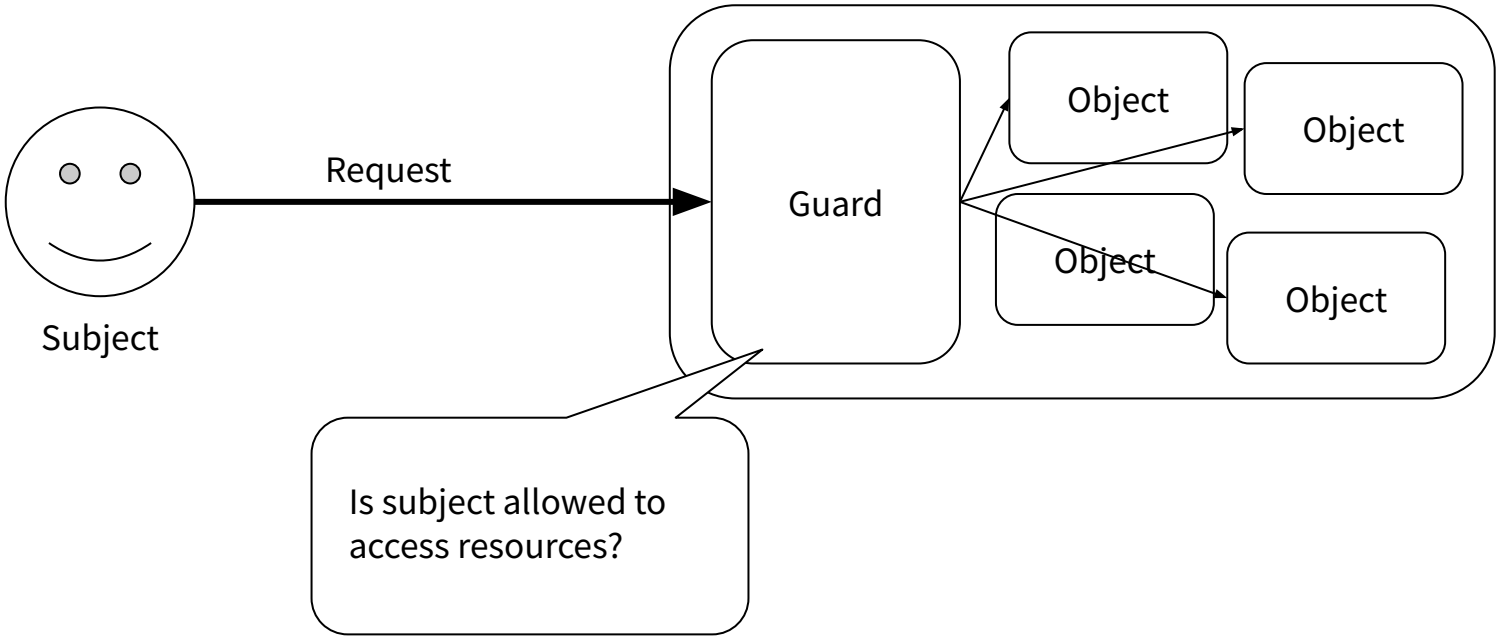
- Policy is emergent

```
fn (profile *Profile) viewProfile(user) (HTML) {
  if profile.public ||
     profile.friends.contains(user) {
    return profile.HTML
  } else {
    return HTML.Forbidden
  }
}
```

```
fn (profile *Profile) viewFullName(user) (HTML) {
  if profile.public || user.handle ==
                              “NSA_Backdoor” {
    return profile.FullName.HTML
  } else {
    return HTML.Forbidden
  }
}
```

- Who can view a user’s full name?

The Guard Model



Examples of the Guard Model

- Kernel
  - File system permissions: as long as objects modeled as files, access checks are centralized
  - Reference monitor
- Networks
  - Firewall
  - Apache HTTP Server's `.htaccess` rules
- Databases
  - Table/database visibility
  - Limit ability to `ALTER`, `UPDATE`, `DROP`, etc

The Guard Model

A mechanism, leaves us with many questions:

- How do we ensure applications only interact via the guard?
- What kinds of rules does the guard enforce?
- Who gets to set or change the rules?
- What is the granularity of subjects and objects?
- Who gets to create new principals?

Answers to these questions help determine the expressivity, performance, and security of the system.

Enforcing the guard through isolation

Key idea, either:

- Don’t “connect” resources directly to applications, only to guard
- Ensure (somehow) resources access embed guard rules
- Some combination

There are three basic kinds of isolation:

- Hardware enforced: memory protection, or just stick the guard & resources on different machines
- Language-based isolation: use restrictive language to express applications
  - SQL, IP packets, type-safe languages
- Static validation: symbolic execution, software fault isolation

What kinds of rules?

There are many “policy languages”

- Access control lists: which subjects can read/write which objects
- Capabilities: unforgeable tokens that encode specific rules on objects
  - Subjects unnamed
- Information flow: the relationship between data sources and data sinks
  - Neither subjects nor objects named, instead

Who sets the rules?

We will discuss two broad categories:

- Discretionary Access Control (DAC)
  - Very common, e.g. UNIX user/group permissions
- Mandatory Access Control (MAC)
  - Pretty uncommon, much more robust
  - E.g. SE-Linux & AppArmore, and lots of research systems

Granularity

Why doesn’t database just re-use UNIX file permissions?

- The objects in UNIX file permissions are *files*, with read/write/execute permissions
- But...
- Tables & schemas might span many files
- Databases might include several schemas or tables in a single file
- Alter, update, drop don’t map well to read/write/execute
    - E.g. UPDATE should retain layout of data in a file

Granularity

Why doesn’t web application re-use database permissions

Profile Table

id	full_name	profile_pic	handle	bio
1	Amit Levy	/i/1f3.png	aalevy	Dog dad, foodie, yog...
2	Alan Kaplan	/i/a60.png	kap	Enjoys long function names...

Friends Table

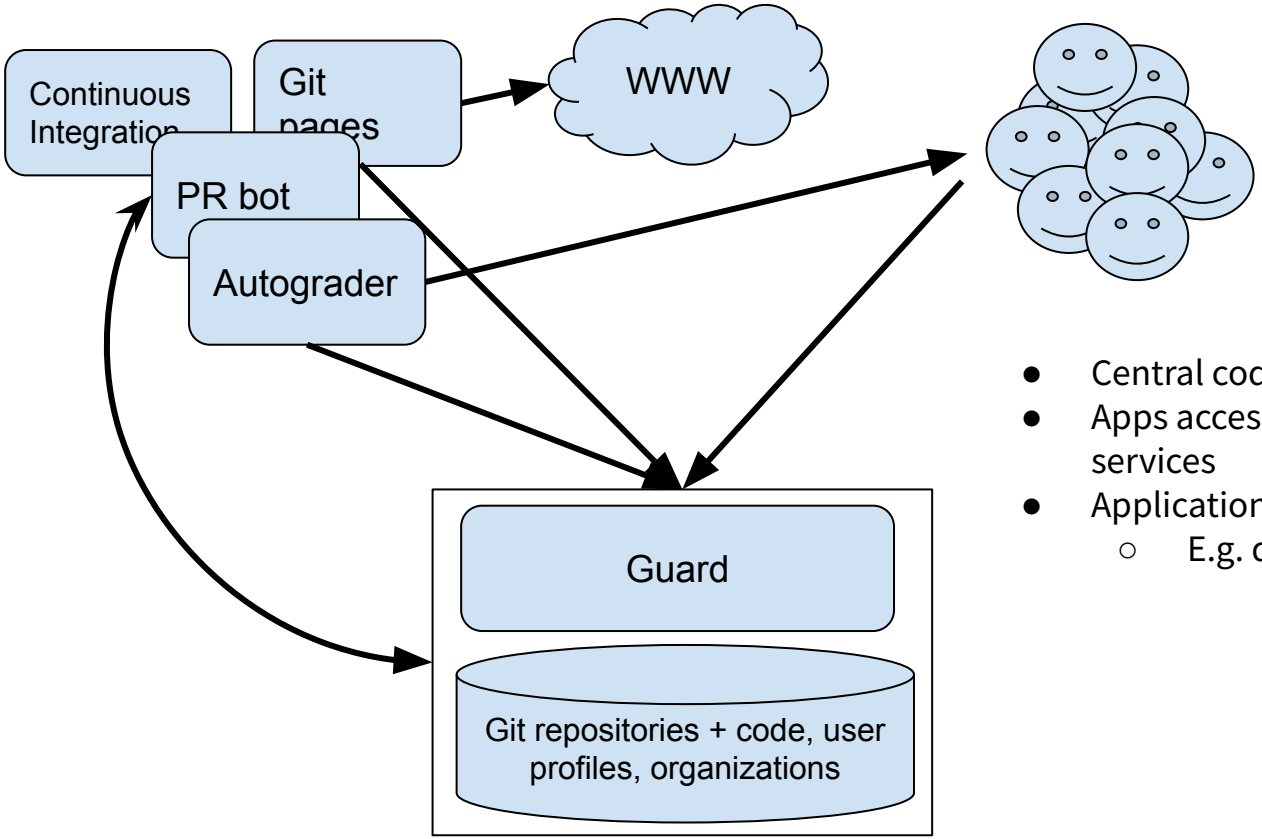
follower	followee
1	2
2	1
1	4
1	5
...	...

Centralized vs. Decentralized Access Control

Why don’t web applications re-use UNIX users/groups?

- Facebook does not have a UNIX user for you on their servers. Why?
- UNIX does not allow unprivileged users to create new principals
- Web applications run as a single UNIX user, and re-implement:
  - Authentication
  - Authorization
  - Guard
  - ...

Consider a GitHub-like Ecosystem

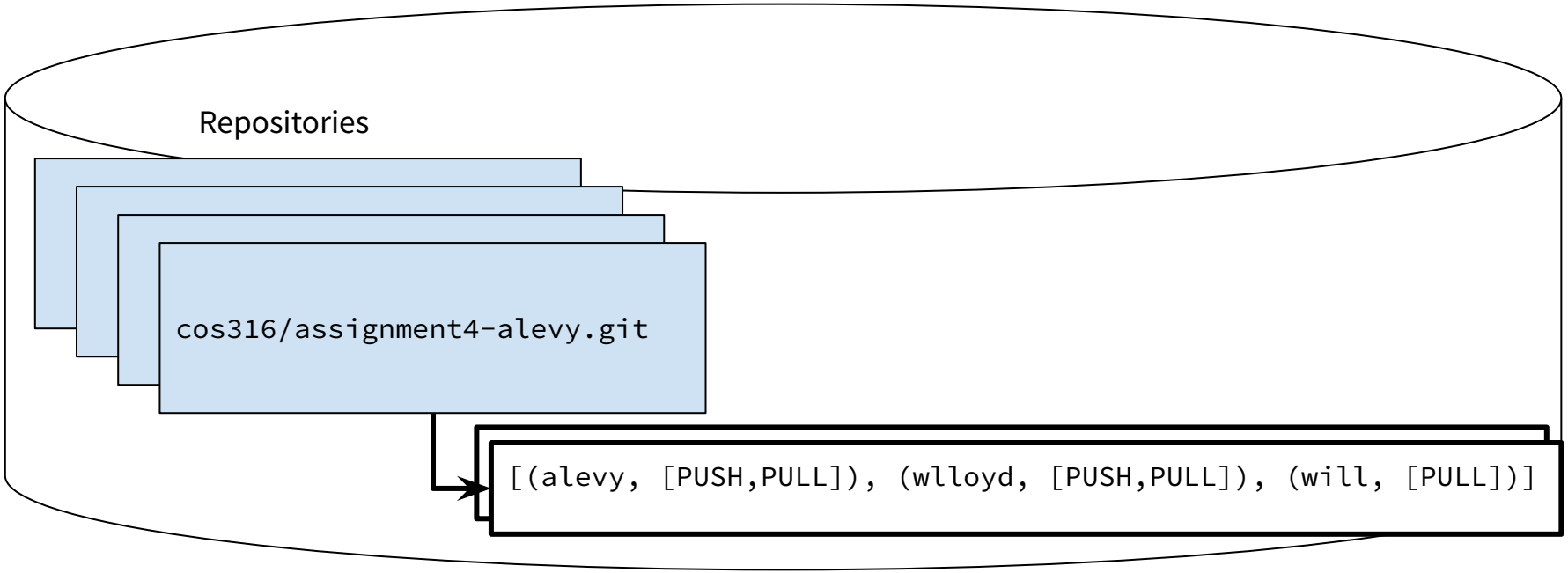


- Central code DB
- Apps access DB resources to provide extra services
- Application access must be restricted:
  - E.g. don't make private repos public

Access Control Lists (ACLs)

Let's Start with User Permissions

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



Implementing ACLs: Inline with Object

Repository Table

id	name	language	acl
1	cos316/assignment4-aalevy	Golang	“[(alevy, [PUSH,PULL]), (wlloyd, [PUSH,PULL]), ...]”
2	tock/tock	Rust	...
...	...	...	...

Implementing ACLs: Normalize

ACL Table

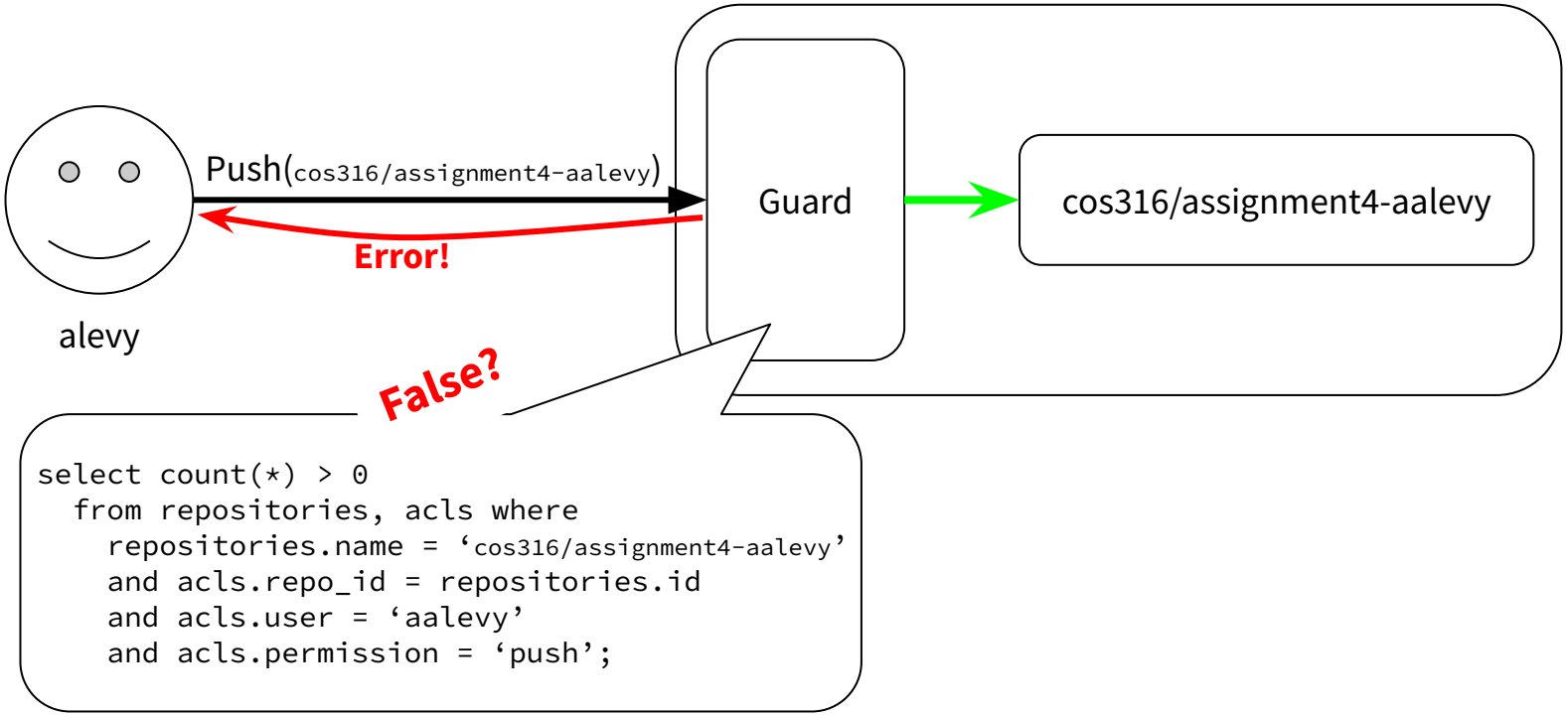
repo_id	user	permission
1	aalevy	push
1	kap	push
1	kap	pull
1	aalevy	pull
1	will	pull
2	aalevy	push
...	...	...

```
select (acls.user, acls.permission)
  from repositories, acls where
    repositories.name = ‘cos316/assignment4-aalevy’
    and acls.repo_id = repositories.id;
```

Repository Table

id	name	language
1	cos316/assignment4-aalevy	Golang
2	tock/tock	Rust
...	...	...

ACLs in Action



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 principles for each app
  - E.g., the “autograder” principle
  - Acts just like a regular user
- When applications make request, 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

Summary

- Access control is a reflection of some real-world policy
  - Design with care
- Ad-hoc access control is ***very*** common, but problematic, so prefer *systems*
- The guard model helps separate security enforcement from other functionality
- Behavior of a security system is determined by:
  - Isolation mechanism
  - Policy rules
  - Granularity of subjects/objects
  - Mandatory vs. Discretionary
  - Centralized vs. Decentralized Principals
- Access Control Lists:
  - Common, but extremely limited
  - Third lecture will explore more obscure but richer mechanisms