User Authentication

Determining if a user is who they say they are before giving them access.
The four means of authenticating user identity are based on:

<table>
<thead>
<tr>
<th>Something the individual knows</th>
<th>Something the individual possesses (token)</th>
<th>Something the individual is (static biometrics)</th>
<th>Something the individual does (dynamic biometrics)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Password, PIN, answers to prearranged questions</td>
<td>• Smartcard, electronic keycard, physical key</td>
<td>• Fingerprint, retina, face</td>
<td>• Voice pattern, handwriting, typing rhythm</td>
</tr>
</tbody>
</table>
Figure 3.2  Multifactor Authentication

Authentication logic using first factor

Authentication logic using second factor

Client

Authentication protocol

Pass

Fail

Client

Authentication protocol

Pass

Fail
The four means of authenticating user identity are based on:

<table>
<thead>
<tr>
<th>Something the individual knows</th>
<th>Something the individual possesses (token)</th>
<th>Something the individual is (static biometrics)</th>
<th>Something the individual does (dynamic biometrics)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Password, PIN, answers to prearranged questions</td>
<td>• Smartcard, electronic keycard, physical key</td>
<td>• Fingerprint, retina, face</td>
<td>• Voice pattern, handwriting, typing rhythm</td>
</tr>
</tbody>
</table>
Password-Based Authentication

- Widely used line of defense against intruders
  - User provides name/login and password
  - System compares password with the one stored for that specified login

- The user ID:
  - Determines that the user is authorized to access the system
  - Determines the user’s privileges
  - Is used in discretionary access control
Hash

• Threat model:
  - Database of password hashes is compromised (happens a LOT)
  - Attacker wants to figure out password

• Hashing:
  - Don’t store the plaintext password, store a hash
  - Compare hashes

• Why?
  - So the attacker can’t just look at the database and see passwords
• **Threat model:**
  - Database of password hashes is compromised (happens a LOT)
  - Attacker wants to figure out password for a given hash

• **Salting:**
  - Add a bit of random stuff ("salt") to password before hashing
  - Random stuff differs per record
  - Store the salt with the hash so we can use it when verifying given passwords

• **Why?**
  - If I hash many possible passwords and find that “c00ldude” hashes to a53d677656e7bcb216b9ef6e38bb7ab1, then anyone with that hash must have that password
  - With a salt, I need to brute-force search *per user* instead of *once-for-everyone*
Iteration count

• Threat model:
  - Database of password hashes is compromised (happens a LOT)
  - Attacker wants to figure out password for a given hash
  - Attacker has lots of fast computers

• Iteration count:
  - Instead of just using $H(data)$, do $H(H(H( ... H(data) ... )))$
  - Increase iteration count to make it very hard for attacker while still being feasible for login checks
  - Makes our hash function “slow” (configurably so!)

• Why?
  - If default hashing has speed of $X$, then an iteration count of 1000 gives a speed of $X/1000$. Login is a tiny amount of time in normal use, but it makes the attacker’s job 1000x harder for very little cost.
Password Vulnerabilities

- **Offline dictionary attack** (e.g., cracking a hashed password)
  - Defense: Make harder by salting, iteration count
- **Specific account attack** (e.g., dictionary attack on account)
  - Defense: Max attempt counter, password complexity requirements
- **Popular password attack** (try few passwords on many accounts)
  - Defense: Password complexity requirements
- **Password guessing against single user** (do research then guess)
  - Defense: User training, password complexity requirements
- **Workstation hijacking** (physically use logged-in workstation)
  - Defense: Physical security, auto-lock timers
- **Exploiting user mistakes** (Post-Its, sharing, unchanged defaults, ...)
  - Defense: Training, single-use expiring passwords for new accounts
- **Exploiting multiple password use**
  - Defense for individual: Password managers with strong crypto
  - Defense for organization: ??????
- **Electronic monitoring** (sniffing network, keylogger, etc.)
  - Defense: Encryption, challenge-response schemes, training
Figure 3.3 UNIX Password Scheme
Evolution of UNIX scheme

- Originally: hash stored in public-readable /etc/passwd file
- Now: hash stored in separate root-readable /etc/shadow file

- Originally: small hash, few iterations
- Later: MD5 hash, more iterations
- Now: SHA 512 hash, configurable iterations
Password Cracking

- **Dictionary attacks**
  - Develop a large dictionary of possible passwords and try each against the password file
  - Each password must be hashed using each salt value and then compared to stored hash values

- **Rainbow table attacks**
  - Pre-compute tables of hash values for all salts
  - A mammoth table of hash values
  - Can be countered by using a sufficiently large salt value and a sufficiently large hash length

- **Password crackers exploit the fact that people choose easily guessable passwords**
  - Shorter password lengths are also easier to crack
Storing passwords correctly

- Storing password plaintext (or encrypted)
- Storing hashed password
- Storing salted hash of password
- Hash function has iteration count
- Just use PBKDF2, scrypt, bcrypt, etc.
- Have a user management library handle it
Where do stolen hashes go?

- Attacker uses directly, sells on black market, or they leak
- Often, eventually, they hit the public internet:
Importance of password storage illustrated (1)

- Plaintext passwords: 100% are “recovered” by attacker (obviously)
- Sorted hashes.org by “percent recovered” – all are unsalted!

<table>
<thead>
<tr>
<th>ID</th>
<th>Name (Algorithm)</th>
<th>#Hashes</th>
<th>Left</th>
<th>Found</th>
<th>Recovered</th>
<th>Updated</th>
</tr>
</thead>
<tbody>
<tr>
<td>780</td>
<td>Pingpong.su</td>
<td>MD5</td>
<td>32'394</td>
<td>0</td>
<td>32'394</td>
<td>100%</td>
</tr>
<tr>
<td>606</td>
<td>Shadi.com</td>
<td>SHA1</td>
<td>1'136'056</td>
<td>35</td>
<td>1'136'056</td>
<td>100%</td>
</tr>
<tr>
<td>35</td>
<td>Zoosk.com</td>
<td>MD5</td>
<td>29'012'254</td>
<td>266</td>
<td>29'012'254</td>
<td>100%</td>
</tr>
<tr>
<td>70</td>
<td>Have I been Pwned V1</td>
<td>SHA1</td>
<td>320'294'448</td>
<td>73'523</td>
<td>320'294'448</td>
<td>99.93%</td>
</tr>
<tr>
<td>26</td>
<td>Op Northkorea</td>
<td>MD5</td>
<td>6'389</td>
<td>4</td>
<td>6'389</td>
<td>99.94%</td>
</tr>
<tr>
<td>698</td>
<td>Fon</td>
<td>MD5</td>
<td>85'033</td>
<td>84</td>
<td>84'949</td>
<td>99.9%</td>
</tr>
</tbody>
</table>

- Scroll to lower percent – almost all are salted.

<table>
<thead>
<tr>
<th>ID</th>
<th>Xronize.com</th>
<th>MD5</th>
<th>43'795</th>
<th>17'106</th>
<th>26'689</th>
<th>60.94%</th>
<th>2018.09.14</th>
</tr>
</thead>
<tbody>
<tr>
<td>733</td>
<td>Politdalforum.com</td>
<td>VBOOLETIN</td>
<td>31'588</td>
<td>12'396</td>
<td>19'192</td>
<td>60.76%</td>
<td>2018.09.01</td>
</tr>
<tr>
<td>115</td>
<td>DayZ.com</td>
<td>MD5</td>
<td>208'236</td>
<td>81'736</td>
<td>126'500</td>
<td>60.75%</td>
<td>2018.05.29</td>
</tr>
<tr>
<td>630</td>
<td>Adultforum.org</td>
<td>VBOOLETIN</td>
<td>7'853</td>
<td>3'094</td>
<td>4'759</td>
<td>60.6%</td>
<td>2018.08.28</td>
</tr>
<tr>
<td>812</td>
<td>Snowandmud.com</td>
<td>VBOOLETIN</td>
<td>53'722</td>
<td>21'259</td>
<td>32'463</td>
<td>60.43%</td>
<td>2018.09.01</td>
</tr>
<tr>
<td>660</td>
<td>Bodyweb.com</td>
<td>VBOOLETIN</td>
<td>79'696</td>
<td>31'800</td>
<td>47'966</td>
<td>60.1%</td>
<td>2018.09.01</td>
</tr>
<tr>
<td>625</td>
<td>Vectorlinux.com</td>
<td>SHA1(SALTPLAIN)</td>
<td>18'343</td>
<td>7'402</td>
<td>10'941</td>
<td>59.65%</td>
<td>2018.05.29</td>
</tr>
</tbody>
</table>
Importance of password storage illustrated (2)

- Scroll to very low percentages...most use bcrypt or similar, which has an iteration count

<table>
<thead>
<tr>
<th>ID</th>
<th>Domain</th>
<th>Algorithm</th>
<th>Cost</th>
<th>Iterations</th>
<th>Complexity</th>
<th>Date</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>971</td>
<td>Forum lightenhope.org</td>
<td>bcrypt</td>
<td>28721</td>
<td>28570</td>
<td>151</td>
<td>0.53%</td>
<td>2018-06-17</td>
</tr>
<tr>
<td>802</td>
<td>Scufgaming.com</td>
<td>WordPress</td>
<td>27809</td>
<td>27801</td>
<td>8</td>
<td>0.26%</td>
<td>2018-05-30</td>
</tr>
<tr>
<td>778</td>
<td>Peesfan.com</td>
<td>VIRUS</td>
<td>426495</td>
<td>425704</td>
<td>701</td>
<td>0.19%</td>
<td>2018-08-28</td>
</tr>
<tr>
<td>558</td>
<td>Dailymotion.com</td>
<td>bcrypt</td>
<td>16147</td>
<td>16139</td>
<td>263</td>
<td>0.05%</td>
<td>2018-06-25</td>
</tr>
<tr>
<td>810</td>
<td>Siriusforum.net</td>
<td>bcrypt</td>
<td>1234</td>
<td>1234</td>
<td>0</td>
<td>0%</td>
<td>2018-05-29</td>
</tr>
<tr>
<td>751</td>
<td>Legion.com</td>
<td>bcrypt</td>
<td>23113</td>
<td>23113</td>
<td>0</td>
<td>0%</td>
<td>2018-05-29</td>
</tr>
<tr>
<td>749</td>
<td>Krotop-gerst.com</td>
<td>bcrypt</td>
<td>27748</td>
<td>27748</td>
<td>0</td>
<td>0%</td>
<td>2018-05-29</td>
</tr>
<tr>
<td>972</td>
<td>Totaljerkface.com</td>
<td>bcrypt</td>
<td>188055</td>
<td>188055</td>
<td>0</td>
<td>0%</td>
<td>2018-08-16</td>
</tr>
</tbody>
</table>

- Conclusion: How you store password has HUGE effect on what happens if (when) they are breached!
Password Selection Strategies

• **User education**
  - Users can be told the importance of using hard to guess passwords and can be provided with guidelines for selecting strong passwords

• **Computer generated passwords**
  - Users have trouble remembering them (good for single-use, bad for long-term)

• **Reactive password checking**
  - System periodically runs its own password cracker to find guessable passwords

• **Complex password policy**
  - User is allowed to select their own password, however the system checks to see if the password is allowable, and if not, rejects it
  - Goal is to eliminate guessable passwords while allowing the user to select a password that is memorable
The four means of authenticating user identity are based on:

<table>
<thead>
<tr>
<th>Something the individual knows</th>
<th>Something the individual possesses (token)</th>
<th>Something the individual is (static biometrics)</th>
<th>Something the individual does (dynamic biometrics)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Password, PIN, answers to prearranged questions</td>
<td>Smartcard, electronic keycard, physical key</td>
<td>Fingerprint, retina, face</td>
<td>Voice pattern, handwriting, typing rhythm</td>
</tr>
</tbody>
</table>
# Types of Cards Used as Tokens

<table>
<thead>
<tr>
<th>Card Type</th>
<th>Defining Feature</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Embossed</td>
<td>Raised characters only, on front</td>
<td>Old credit card</td>
</tr>
<tr>
<td>Magnetic stripe</td>
<td>Magnetic bar on back, characters on front</td>
<td>Bank card</td>
</tr>
<tr>
<td>Memory</td>
<td>Electronic memory inside</td>
<td>Prepaid phone card</td>
</tr>
<tr>
<td>Smart Contact</td>
<td>Electronic memory and processor inside</td>
<td>Biometric ID card</td>
</tr>
<tr>
<td>Contactless</td>
<td>Electrical contacts exposed on surface</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Radio antenna embedded inside</td>
<td></td>
</tr>
</tbody>
</table>
Memory Cards

- Can store but do not process data
- The most common is the magnetic stripe card
- Can include an internal electronic memory
- Can be used alone for physical access
  - Hotel room
  - ATM
- Provides significantly greater security when combined with a password or PIN
- Drawbacks of memory cards include:
  - Requires a special reader
  - Loss of token
  - User dissatisfaction
Smart Tokens

• **Physical characteristics:**
  - Include an embedded microprocessor
  - A smart token that looks like a bank card
  - Can look like calculators, keys, small portable objects

• **User interface:**
  - Manual interfaces include a keypad and display for human/token interaction

• **Electronic interface**
  - A smart card or other token requires an electronic interface to communicate with a compatible reader/writer
  - Contact and contactless interfaces

• **Authentication protocol:**
  - Classified into three categories:
    • Static
    • Dynamic password generator
    • Challenge-response
Smart Cards

• **Most important category of smart token**
  o Has the appearance of a credit card
  o Has an electronic interface
  o May use any of the smart token protocols

• **Contain:**
  o An entire microprocessor
    • Processor
    • Memory
    • I/O ports

• **Typically include three types of memory:**
  o Read-only memory (ROM)
    • Stores data that does not change during the card’s life
  o Electrically erasable programmable ROM (EEPROM)
    • Holds application data and programs
  o Random access memory (RAM)
    • Holds temporary data generated when applications are executed
Smart Card Activation

ATR

Protocol negotiation PTS

Negotiation Answer PTS

Command APDU

Response APDU

End of Session

APDU = application protocol data unit
ATR = Answer to reset
PTS = Protocol type selection

Figure 3.6 Smart Card/Reader Exchange
The four means of authenticating user identity are based on:

<table>
<thead>
<tr>
<th>Something the individual knows</th>
<th>Something the individual possesses (token)</th>
<th>Something the individual is (static biometrics)</th>
<th>Something the individual does (dynamic biometrics)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Password, PIN, answers to prearranged questions</td>
<td>Smartcard, electronic keycard, physical key</td>
<td>Fingerprint, retina, face</td>
<td>Voice pattern, handwriting, typing rhythm</td>
</tr>
</tbody>
</table>
Biometric Authentication

• Attempts to authenticate an individual based on unique physical characteristics
• Based on pattern recognition
• Is technically complex and expensive when compared to passwords and tokens
• Physical characteristics used include:
  o Facial characteristics
  o Fingerprints
  o Hand geometry
  o Retinal pattern
  o Iris
  o Signature
  o Voice
Figure 3.8 Cost Versus Accuracy of Various Biometric Characteristics in User Authentication Schemes.
Figure 3.9 A Generic Biometric System. Enrollment creates an association between a user and the user's biometric characteristics. Depending on the application, user authentication either involves verifying that a claimed user is the actual user or identifying an unknown user.
Figure 3.10 Profiles of a Biometric Characteristic of an Imposter and an Authorized Users. In this depiction, the comparison between presented feature and a reference feature is reduced to a single numeric value. If the input value ($s$) is greater than a preassigned threshold ($t$), a match is declared.
Figure 3.11  Idealized Biometric Measurement
Operating Characteristic Curves (log-log scale)
Figure 3.12  Actual Biometric Measurement Operating Characteristic Curves, reported in [MANS01]. To clarify differences among systems, a log-log scale is used.
Remote User Authentication

• Authentication over a network, the Internet, or a communications link is more complex

• Additional security threats such as:
  - Eavesdropping, capturing a password, replaying an authentication sequence that has been observed

• Generally rely on some form of a challenge-response protocol to counter threats
Challenge-Response scheme

• Assume we have some authentication secret S
  ▪ Token value, biometric signature, etc...
• Don’t want to send it (or even its hash!)
• Instead, server issues a challenge (random value \(R\)) to client that can only be answered if it has S, but which doesn’t reveal S.

Client

I’m user Bob

Oh yeah? Assume \(R=5248\), so compute \(h(R + h(S))\) for me, where S is Bob’s secret.

Here’s \(h(R + h(S))\)

Server

oh ok cool
Challenge-Response: What about passwords?

- In the scheme shown, if the password hash is leaked, it’s *equivalent* to having the actual password, since we only need $h(S)$!
- Other challenge-response schemes avoid this issue, e.g. **Salted Challenge Response Authentication Mechanism (SCRAM)**

**Communications sequence**

```
<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>username, ClientNonce</td>
<td>ClientKey</td>
</tr>
<tr>
<td>salt, iteration, CombinedNonce</td>
<td>StoredKey</td>
</tr>
<tr>
<td>ClientProof, CombinedNonce</td>
<td>ClientProof</td>
</tr>
<tr>
<td>ServerSignature</td>
<td></td>
</tr>
</tbody>
</table>
```

**Mutations done to the salted password**

SaltedPassword

- ClientKey = HMAC(SaltedPassword, "Client Key")
- StoredKey = $H$(ClientKey)
- ServerKey = HMAC(SaltedPassword, "Server Key")

Auth = {username, salt, iteration, CombinedNonce}

- ClientProof = ClientKey ^ HMAC(StoredKey, Auth)
- ServerProof = HMAC(ServerKey, Auth)

For more, see [Wikipedia](https://en.wikipedia.org/wiki/SCRAM) or [this article](https://example.com/articles/scrampassword)
### Table 3.5

Some Potential Attacks, Susceptible Authenticators, and Typical Defenses

<table>
<thead>
<tr>
<th>Attacks</th>
<th>Authenticators</th>
<th>Examples</th>
<th>Typical defenses</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Client attack</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Password</td>
<td>Password</td>
<td>Guessing, exhaustive search</td>
<td>Large entropy; limited attempts</td>
</tr>
<tr>
<td>Token</td>
<td>Token</td>
<td>Exhaustive search</td>
<td>Large entropy; limited attempts, theft of object requires presence</td>
</tr>
<tr>
<td>Biometric</td>
<td>Biometric</td>
<td>False match</td>
<td>Large entropy; limited attempts</td>
</tr>
<tr>
<td><strong>Host attack</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Password</td>
<td>Password</td>
<td>Plaintext theft, dictionary/exhaustive search</td>
<td>Hashing; large entropy; protection of password database</td>
</tr>
<tr>
<td>Token</td>
<td>Token</td>
<td>Passcode theft</td>
<td>Same as password; 1-time passcode</td>
</tr>
<tr>
<td>Biometric</td>
<td>Biometric</td>
<td>Template theft</td>
<td>Capture device authentication; challenge response</td>
</tr>
<tr>
<td><strong>Eavesdropping, theft, and copying</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Password</td>
<td>Password</td>
<td>&quot;Shoulder surfing&quot;</td>
<td>User diligence to keep secret; administrator diligence to quickly revoke compromised passwords; multifactor authentication</td>
</tr>
<tr>
<td>Token</td>
<td>Token</td>
<td>Theft, counterfeiting hardware</td>
<td>Multifactor authentication; tamper resistant/evident token</td>
</tr>
<tr>
<td>Biometric</td>
<td>Biometric</td>
<td>Copying (spoofing) biometric</td>
<td>Copy detection at capture device and capture device authentication</td>
</tr>
<tr>
<td><strong>Replay</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Password</td>
<td>Password</td>
<td>Replay stolen password response</td>
<td>Challenge-response protocol</td>
</tr>
<tr>
<td>Token</td>
<td>Token</td>
<td>Replay stolen passcode response</td>
<td>Challenge-response protocol; 1-time passcode</td>
</tr>
<tr>
<td>Biometric</td>
<td>Biometric</td>
<td>Replay stolen biometric template response</td>
<td>Copy detection at capture device and capture device authentication via challenge-response protocol</td>
</tr>
<tr>
<td><strong>Trojan horse</strong></td>
<td>Password, token, biometric</td>
<td>Installation of rogue client or capture device</td>
<td>Authentication of client or capture device within trusted security perimeter</td>
</tr>
<tr>
<td><strong>Denial of service</strong></td>
<td>Password, token, biometric</td>
<td>Lockout by multiple failed authentications</td>
<td>Multifactor with token</td>
</tr>
</tbody>
</table>

(Table is on page 96 in the textbook)
Access control

So you’ve proven who you are, but what are you allowed to do?
• Core concepts
• Access control policies:
  ▪ DAC
    • UNIX file system
  ▪ MAC
  ▪ RBAC
  ▪ ABAC
• Identity federation
### Subjects, Objects, Actions, and Rights

<table>
<thead>
<tr>
<th>Subject (initiator)</th>
<th>Verb (request)</th>
<th>Right (permission)</th>
<th>Object (target)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• The thing making the request (e.g. the user)</td>
<td>• The operation to perform (e.g., read, delete, etc.)</td>
<td>• A specific ability for the subject to do the action to the object.</td>
<td>• The thing that’s being hit by the request (e.g., a file).</td>
</tr>
</tbody>
</table>
Categories of Access Control Policies

• **Discretionary AC (DAC):** There’s a list of permissions attached to the subject or object (or possibly a giant heap of global rules).

• **Mandatory AC (MAC):** Objects have classifications, subjects have clearances, subjects cannot give additional permissions.
  - An overused/abused term

• **Role-Based AC (RBAC):** Subjects belong to roles, and roles have all the permissions.
  - The current Enterprise IT buzzword meaning “good” security

• **Attribute-Based AC (ABAC):** Subjects and objects have attributes, rules engine applies predicates to these to determine access
  - Allows fine-grained expression
  - Usually complex, seldom implemented
Discretionary Access Control (DAC)

• Scheme in which an entity may enable another entity to access some resource
• Often provided using an access matrix
  o One dimension consists of identified subjects that may attempt data access to the resources
  o The other dimension lists the objects that may be accessed
• Each entry in the matrix indicates the access rights of a particular subject for a particular object
bool IsActionAllowed(subject, object, action) {
    if (action ∈ get_permissions(subject, object))
        return true
}

DAC model
Implementation

- Can use various data structures, none of which should surprise you

---

**Flat list**

<table>
<thead>
<tr>
<th>Subject</th>
<th>Access Mode</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Own</td>
<td>File 1</td>
</tr>
<tr>
<td>A</td>
<td>Read</td>
<td>File 1</td>
</tr>
<tr>
<td>A</td>
<td>Write</td>
<td>File 1</td>
</tr>
<tr>
<td>A</td>
<td>Own</td>
<td>File 3</td>
</tr>
<tr>
<td>A</td>
<td>Read</td>
<td>File 3</td>
</tr>
<tr>
<td>A</td>
<td>Write</td>
<td>File 3</td>
</tr>
<tr>
<td>B</td>
<td>Read</td>
<td>File 1</td>
</tr>
<tr>
<td>B</td>
<td>Own</td>
<td>File 2</td>
</tr>
<tr>
<td>B</td>
<td>Read</td>
<td>File 2</td>
</tr>
<tr>
<td>B</td>
<td>Write</td>
<td>File 2</td>
</tr>
<tr>
<td>B</td>
<td>Write</td>
<td>File 3</td>
</tr>
<tr>
<td>B</td>
<td>Read</td>
<td>File 4</td>
</tr>
<tr>
<td>C</td>
<td>Read</td>
<td>File 1</td>
</tr>
<tr>
<td>C</td>
<td>Write</td>
<td>File 1</td>
</tr>
<tr>
<td>C</td>
<td>Read</td>
<td>File 2</td>
</tr>
<tr>
<td>C</td>
<td>Own</td>
<td>File 4</td>
</tr>
<tr>
<td>C</td>
<td>Read</td>
<td>File 4</td>
</tr>
<tr>
<td>C</td>
<td>Write</td>
<td>File 4</td>
</tr>
</tbody>
</table>

---

**Linked list**

- File 1: A (Own, Read, Write) \(\rightarrow\) B (Own, Read) \(\rightarrow\) C (Own, Write)
- File 2: B (Own, Read, Write) \(\rightarrow\) C (Own, Read, Write)
- File 3: A (Own, Read, Write) \(\rightarrow\) B (Own, Read, Write) \(\rightarrow\) C (Own, Read, Write)
- File 4: B (Own, Read) \(\rightarrow\) C (Own, Read, Write)

---

**Matrix**

- User A: File 1 (Own Read Write) \(\rightarrow\) File 2 (Own Read Write) \(\rightarrow\) File 3 (Own Read Write) \(\rightarrow\) File 4 (Own Read Write)
- User B: Read \(\rightarrow\) File 2 (Own Read Write) \(\rightarrow\) Read \(\rightarrow\) File 4 (Own Read Write)
- User C: Read \(\rightarrow\) File 3 (Read) \(\rightarrow\) Own Read Write

---

**Figure 4.2 Example of Access Control Structures**

- **Access control matrix**
- **Linked list**
- **Capability lists**

---
UNIX File Access Control

UNIX files are administered using inodes (index nodes)

• Control structures with key information needed for a particular file
• Several file names may be associated with a single inode
• An active inode is associated with exactly one file
• File attributes, permissions and control information are sorted in the inode
• On the disk there is an inode table, or inode list, that contains the inodes of all the files in the file system
• When a file is opened its inode is brought into main memory and stored in a memory resident inode table

Directories are structured in a hierarchical tree

• May contain files and/or other directories
• Contains file names plus pointers to associated inodes
UNIX
File Access Control

- Unique user identification number (user ID)
- Member of a primary group identified by a group ID
- Belongs to a specific group
- 12 protection bits
  - Specify read, write, and execute permission for the owner of the file, members of the group and all other users
- The owner ID, group ID, and protection bits are part of the file’s inode

(a) Traditional UNIX approach (minimal access control list)
Traditional UNIX File Access Control

- "Set user ID" (SetUID)
- "Set group ID" (SetGID)
  - System temporarily uses rights of the file owner/group in addition to the real user’s rights when making access control decisions
  - Enables privileged programs to access files/resources not generally accessible
- Sticky bit
  - When applied to a directory it specifies that only the owner of any file in the directory can rename, move, or delete that file
- Superuser
  - Is exempt from usual access control restrictions
  - Has system-wide access
Figure 4.5   UNIX File Access Control
(a) Traditional UNIX approach (minimal access control list)
rw- r-- ---
Owner class
Group class
Other class
user: :rw-
user:joe:rw-
group::r--
mask::rw-
other::---

(b) Extended access control list

Figure 4.5   UNIX File Access Control
File system access control lists (ACLs)

- Arbitrary list of rules governing access per-file/directory
- More flexible than classic UNIX permissions, but more metadata to store/check

Windows ACL UI

Examples of Linux ACL commands

From Arch Wiki
Topics

• Core concepts

• Access control policies:
  ▪ DAC
  ▪ UNIX file system
  ▪ MAC
  ▪ RBAC
  ▪ ABAC

• Identity federation
MAC example: SELinux

- Developed by U.S. Dept of Defense
- General deployment starting 2003
- Can apply rules to virtually every user/process/hardware pair
- Rules are governed by system administrator only
  - No such thing as “selinux_chmod” for users
MAC example: SELinux
MAC model

```csharp
bool IsActionAllowed(subject, object, action) {
    for each rule in rules:
        if rule allows (subject, object, action) return true
    return false
}
```
• Core concepts

• Access control policies:
  ▪ DAC
  ▪ UNIX file system
  ▪ MAC
  ▪ RBAC
  ▪ ABAC

• Identity federation
RBAC: The thing you invent if you spend enough time doing access control

- Scenario:
  - Frank: “Bob just got hired, please given him access.”
  - Admin: “What permissions does he need?”
  - Frank: “Same as me.”

- Later, a new system is added
  - Bob: “Why can’t I access the new system?!”
  - Admin: “Oh, I didn’t know you needed it too…”
  - Bob: “I need everything Frank has!”

- Later, Frank is promoted to CTO
  - Admin: “Welp, looks like Bob also needs access to our private earnings, since this post-it says he gets everything Frank has…”

- The admin is later fired amidst allegations of conspiracy to commit insider trading with Bob. He dies in prison. 😞
Figure 4.6 Users, Roles, and Resources
• Decide what KINDS of users you have (roles)
• Assign permission to roles.
• Assign users to roles.

• When a role changes, everyone gets the change.
• When a user’s role changes, that user gets a whole new set of permissions.
• No more special unique snowflakes.

• Roles may be partially ordered, e.g. “Production developer” inherits from “Developer” and adds access to the production servers.
RBAC implementation

- Unsurprisingly, you can represent this using various data structures.
  - Anything that can represent two matrices:

![Access Control Matrix Representation of RBAC](image)

Figure 4.7  Access Control Matrix Representation of RBAC
bool IsActionAllowed(subject, object, action) {
    if (action ∈ get_permissions(subject.role, object))
        return true
}

RBAC model
Figure 4.9 Example of Role Hierarchy
Constraints - RBAC

- Provide a means of adapting RBAC to the specifics of administrative and security policies of an organization
- A defined relationship among roles or a condition related to roles

Types:

<table>
<thead>
<tr>
<th>Mutually exclusive roles</th>
<th>Cardinality</th>
<th>Prerequisite roles</th>
</tr>
</thead>
<tbody>
<tr>
<td>A user can only be assigned to one role in the set (either during a session or statically)</td>
<td>Setting a maximum number with respect to roles</td>
<td>Dictates that a user can only be assigned to a particular role if it is already assigned to some other specified role</td>
</tr>
<tr>
<td>Any permission (access right) can be granted to only one role in the set</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Topics

• Core concepts

• Access control policies:
  ▪ DAC
    ● UNIX file system
  ▪ MAC
  ▪ RBAC
  ▪ ABAC

• Identity federation
Attribute-Based Access Control (ABAC)

• Authorizations based on conditions on properties of both the resource and the subject
• Strength is its flexibility and expressive power
• Main obstacle: complexity to administer (and understand)

I have no evidence that any non-academic person has ever used this, so we’re skipping it. Slides included in case you care to learn more.
ABAC Model: Attributes

**Subject attributes**
- A subject is an active entity that causes information to flow among objects or changes the system state
- Attributes define the identity and characteristics of the subject

**Object attributes**
- An object (or resource) is a passive information system-related entity containing or receiving information
- Objects have attributes that can be leverages to make access control decisions

**Environment attributes**
- Describe the operational, technical, and even situational environment or context in which the information access occurs
- These attributes have so far been largely ignored in most access control policies
ABAC in a nutshell

POLICY

Yes or no
ABAC

Distinguishable because it controls access to objects by evaluating rules against the attributes of entities, operations, and the environment relevant to a request.

Relies upon the evaluation of attributes of the subject, attributes of the object, and a formal relationship or access control rule defining the allowable operations for subject-object attribute combinations in a given environment.

Systems are capable of enforcing DAC, RBAC, and MAC concepts.

Allows an unlimited number of attributes to be combined to satisfy any access control rule.
Figure 4.10  Simple ABAC Scenario

Access Control Policy

Environmental Conditions

Rules

Decision

Enforce

Subject Attributes

Subject

Object Attributes

Object

Subject

Name

Clearance

Etc.

Affiliation

Type

Owner

Etc.

Classification
bool IsActionAllowed(subject, object, action) {
    for each rule in rules {
        // The rule is basically code that examines all attributes of subject and object as well as the global environment; the rule is highly expressive, and so could basically do anything. If it says yes, return true
    }
    return false
}
Topics

• Core concepts

• Access control policies:
  ▪ DAC
    ▪ UNIX file system
  ▪ MAC
  ▪ RBAC
  ▪ ABAC

• Identity federation
Identity Federation

• Term used to describe the technology, standards, policies, and processes that allow an organization to trust digital identities, identity attributes, and credentials created and issued by another organization

• Addresses two questions:
  o How do you trust identities of individuals from external organizations who need access to your systems
  o How do you vouch for identities of individuals in your organization when they need to collaborate with external organizations
Identity Federation made simple

- Translation:
  - Corporate providers: Google/Facebook
  - Open provider framework: OpenID

- Allow one entity to manage the concept of “logging in” (credentials, etc.), and communicate that to another entity on behalf of the user

- Want a standard to support federation from any provider? OAuth
- Duke has an authentication system: Duke NetID
  - You can write apps that use OAuth to allow login via Duke NetID
Any questions?