ECE560
Computer and Information Security

Fall 2023

User Authentication and Access Control

Tyler Bletsch

Duke University

User Authentication

Determining if a user is who they say they are before giving them access.

Four means of authentication

Something you **know**	- Password, PIN, answers to prearranged questions
Something you **have** (*token*)	- Smartcard, electronic keycard, physical key
Something you **are** (*static biometrics*)	- Fingerprint, retina, face
Something you **do** (*dynamic biometrics*)	- Voice pattern, handwriting

Passwords

- Most common authentication mechanism
  - User provides username and password, must match server records (For reference, see every computer thing ever)

- The hard parts:
  - How to **store** passwords?
  - How to **communicate** passwords? (covered later on)

Storing passwords: Hashing

- Given setup: store passwords in plaintext
- Threat model:
  - Database of user info is compromised (happens a LOT)
  - Attacker wants to figure out password
- Attack:
  - Attacker just looks at the database and sees the passwords
- Improvement: **Hashing**
  - Don’t store the plaintext password, store a hash
  - Compare hashes

Storing passwords: Salting

- Given setup: store hashed passwords
- Threat model:
  - Database of password hashes is compromised (happens a LOT)
  - Attacker wants to figure out password for a given hash
- Attack:
  - Attacker hashes many possible passwords and finds that “c00ldude” hashes to a53d677656e7bcb216b9ef6e38bb7ab1. *Anyone* with that hash must have that password!
    - Can also see users with the same password, even if it’s unknown!
- Improvement: **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
- Result: I need to brute-force search *per user* instead of once for *everyone*

Storing passwords: Iteration count

- Given setup: Store salted hashed passwords
- 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
- Attack
  - Okay, given the salt for a specific user, I do hash a billion possibilities; still often likely to find a match!
- Improvement: **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

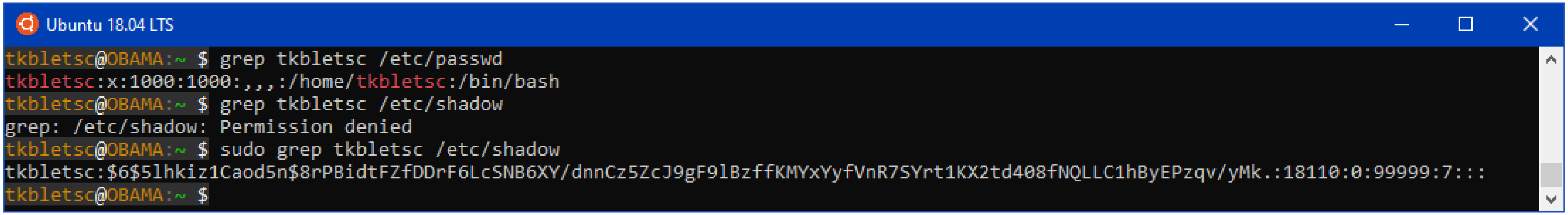
- **Offline dictionary attack**: Crack a hashed password
  - Defense: Make harder by salting, iteration count
- **Online dictionary attack**: Try dictionary logins to actual live system
  - Defense: Max attempt counter, password complexity requirements
- **Password spraying**: Try few common passwords on many accounts/systems
  - Defense: Password complexity requirements
- **Credential stuffing**: Try the same user+password many places (often the creds are leaked from a prior breach)
  - Defense for individual: Password managers with strong crypto
  - Defense for organization: ?????
- **Password guessing**: Do research then guess
  - Defense: User training, password complexity requirements
- **Exploiting user mistakes**: Post-It notes, sharing, unchanged defaults, etc.
  - Defense: Training, single-use expiring passwords for new accounts
- **Electronic monitoring**: Sniffing network, installing keylogger, etc.
  - Defense: Encryption, challenge-response schemes, training

E.g.: Trump's Twitter password was guessed. It was "maga2020!" 😐

UNIX password scheme

- Originally: hash stored in public-readable /etc/passwd file
  - Hashes were public; relied entirely on them being hard to crack
  - People slowly figured out in the 80s this was feasible
    (god what an awesome/lazy time to be an attacker…)
- 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



```
tkbletsc@OBAMA:~ $ grep tkbletsc /etc/passwd
tkbletsc:x:1000:1000:,,,:/home/tkbletsc:/bin/bash
tkbletsc@OBAMA:~ $ grep tkbletsc /etc/shadow
grep: /etc/shadow: Permission denied
tkbletsc@OBAMA:~ $ sudo grep tkbletsc /etc/shadow
tkbletsc:$6$5lhkiz1Caod5n$8rPBidtFZfDDrF6LcSNB6XY/dnnCz5ZcJ9gF9lBzffKMYxYyfVnR7SYrt1KX2td408fNQLLC1hByEPzqv/yMk.:18110:0:99999:7:::
tkbletsc@OBAMA:~ $
```

Passwords normally changed with `passwd` tool
Can generate shadow-compatible hash strings with `mkpasswd`

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)

T-Mobile Link

SONY Link

Adobe Link

- Storing hashed password

Windows Link

LinkedIn Link

- Storing salted hash of password

DISQUS Link

- Hash function has iteration count

I couldn't find anyone who bothered to do this yet didn't just use one of the functions below

- Just use PBKDF2, scrypt, bcrypt, etc.

Dropbox Link

Linux Link

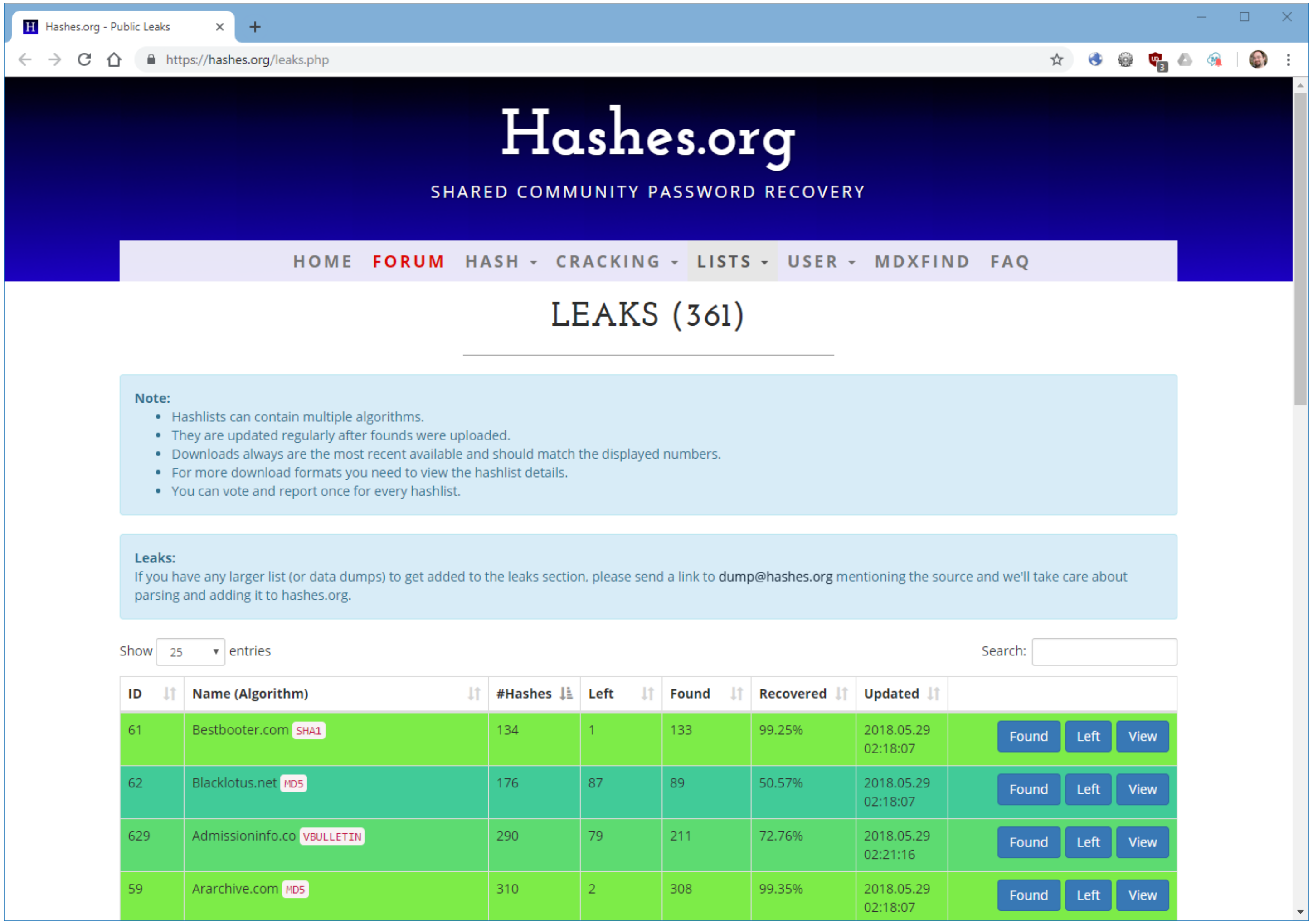
- Have a user management library handle it

django

node JS

Where do stolen hashes go?

- Attacker uses directly, sells on black market, or they leak
- Often, eventually, they hit the public internet:



Hashes.org

SHARED COMMUNITY PASSWORD RECOVERY

LEAKS (361)

Note:

- Hashlists can contain multiple algorithms.
- They are updated regularly after founds were uploaded.
- Downloads always are the most recent available and should match the displayed numbers.
- For more download formats you need to view the hashlist details.
- You can vote and report once for every hashlist.

Leaks:
If you have any larger list (or data dumps) to get added to the leaks section, please send a link to dump@hashes.org mentioning the source and we'll take care about parsing and adding it to hashes.org.

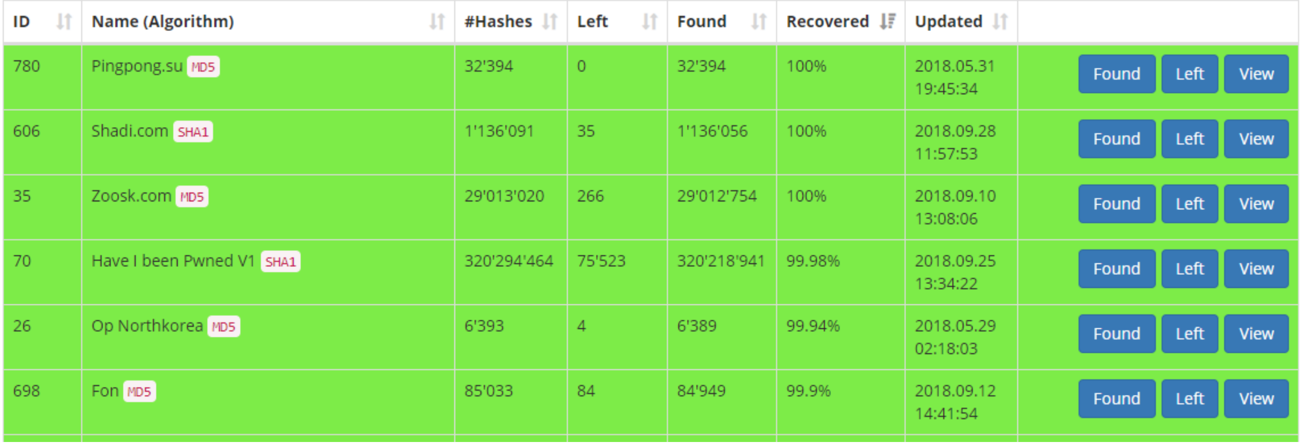
Show 25 entries

Search:

ID	Name (Algorithm)	#Hashes	Left	Found	Recovered	Updated	
61	Bestbooter.com SHA1	134	1	133	99.25%	2018.05.29 02:18:07	Found Left View
62	Blacklotus.net MD5	176	87	89	50.57%	2018.05.29 02:18:07	Found Left View
629	Admissioninfo.co VBULLETIN	290	79	211	72.76%	2018.05.29 02:21:16	Found Left View
59	Ararchive.com MD5	310	2	308	99.35%	2018.05.29 02:18:07	Found Left View

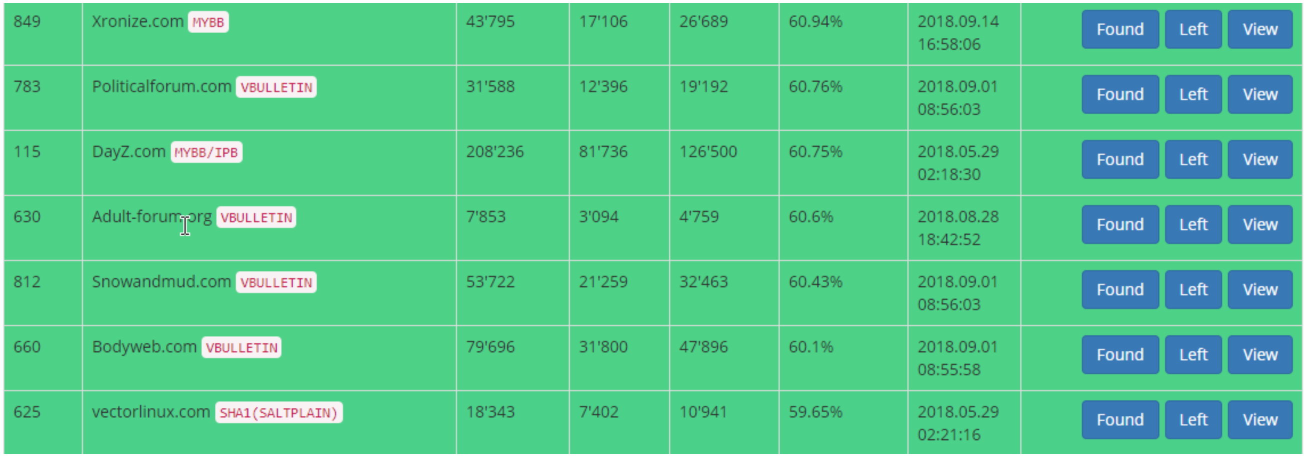
Importance of password storage illustrated (1)

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



ID	Name (Algorithm)	#Hashes	Left	Found	Recovered	Updated	
780	Pingpong.su MD5	32'394	0	32'394	100%	2018.05.31 19:45:34	Found Left View
606	Shadi.com SHA1	1'136'091	35	1'136'056	100%	2018.09.28 11:57:53	Found Left View
35	Zoosk.com MD5	29'013'020	266	29'012'754	100%	2018.09.10 13:08:06	Found Left View
70	Have I been Pwned V1 SHA1	320'294'464	75'523	320'218'941	99.98%	2018.09.25 13:34:22	Found Left View
26	Op Northkorea MD5	6'393	4	6'389	99.94%	2018.05.29 02:18:03	Found Left View
698	Fon MD5	85'033	84	84'949	99.9%	2018.09.12 14:41:54	Found Left View

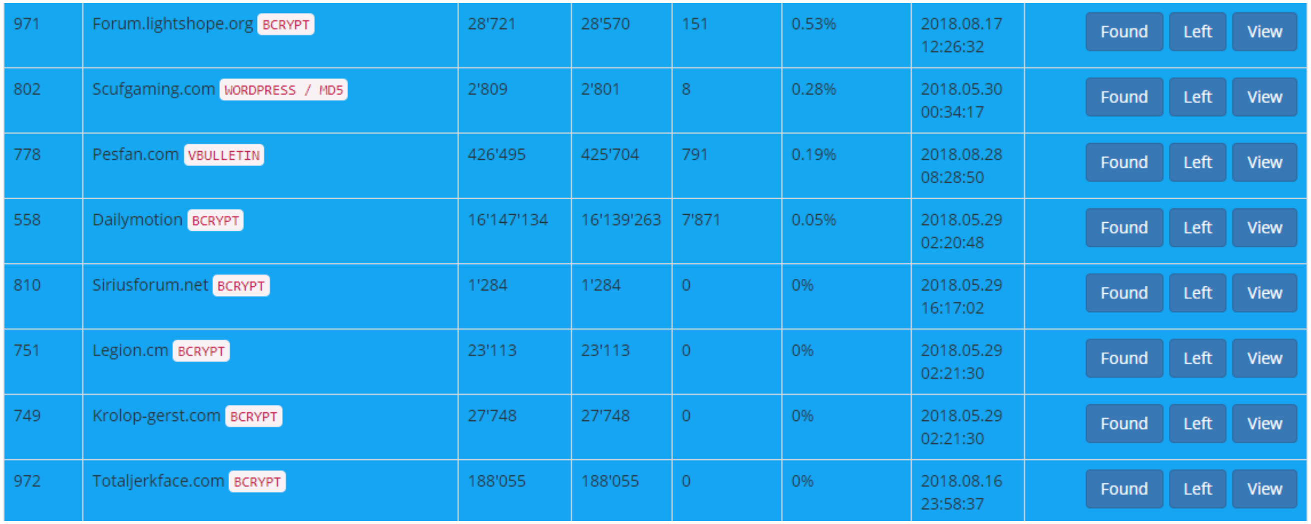
- Scroll to lower percent – almost all are salted.



849	Xronize.com MYBB	43'795	17'106	26'689	60.94%	2018.09.14 16:58:06	Found Left View
783	Politicalforum.com VBULLETIN	31'588	12'396	19'192	60.76%	2018.09.01 08:56:03	Found Left View
115	DayZ.com MYBB/IPB	208'236	81'736	126'500	60.75%	2018.05.29 02:18:30	Found Left View
630	Adult-forum.org VBULLETIN	7'853	3'094	4'759	60.6%	2018.08.28 18:42:52	Found Left View
812	Snowandmud.com VBULLETIN	53'722	21'259	32'463	60.43%	2018.09.01 08:56:03	Found Left View
660	Bodyweb.com VBULLETIN	79'696	31'800	47'896	60.1%	2018.09.01 08:55:58	Found Left View
625	vectorlinux.com SHA1(SALTPLAIN)	18'343	7'402	10'941	59.65%	2018.05.29 02:21:16	Found Left View

Importance of password storage illustrated (2)

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



971	Forum.lightshope.org BCRYPT	28'721	28'570	151	0.53%	2018.08.17 12:26:32	Found Left View
802	Scufgaming.com WORDPRESS / MD5	2'809	2'801	8	0.28%	2018.05.30 00:34:17	Found Left View
778	Pesfan.com VBULLETIN	426'495	425'704	791	0.19%	2018.08.28 08:28:50	Found Left View
558	Dailymotion BCRYPT	16'147'134	16'139'263	7'871	0.05%	2018.05.29 02:20:48	Found Left View
810	Siriusforum.net BCRYPT	1'284	1'284	0	0%	2018.05.29 16:17:02	Found Left View
751	Legion.cm BCRYPT	23'113	23'113	0	0%	2018.05.29 02:21:30	Found Left View
749	Krolop-gerst.com BCRYPT	27'748	27'748	0	0%	2018.05.29 02:21:30	Found Left View
972	Totaljerkface.com BCRYPT	188'055	188'055	0	0%	2018.08.16 23:58:37	Found Left View

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

Password Selection Strategies

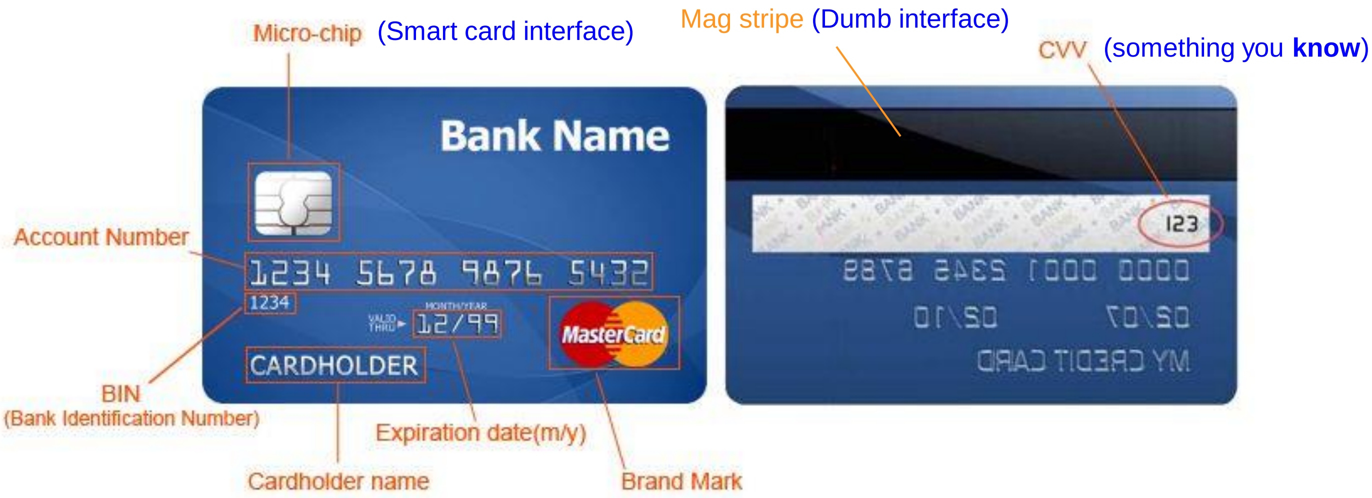
- **User education**
  - Users can be told the importance of using hard to guess passwords and can be provided with guidelines for selecting strong passwords
- **Computer generated passwords**
  - Users have trouble remembering them
    (good for single-use, bad for long-term)
- **Reactive password checking**
  - 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

Four means of authentication

Something you **know**	• Password, PIN, answers to prearranged questions
Something you **have** (*token*)	• Smartcard, electronic keycard, physical key
Something you **are** (*static biometrics*)	• Fingerprint, retina, face
Something you **do** (*dynamic biometrics*)	• Voice pattern, handwriting

Types of tokens (1)

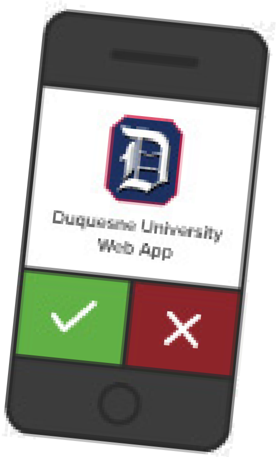
- **Cards** (or card-like things)
  - Magnetic stripe (read-only, clear communication)
  - Memory card (read-only/read-write, no processor, clear communication)
  - **Smart card** (read-only/read-write, has processor, encrypted communication)
    - May be **contact** (e.g., this bank card) or **contactless** (e.g., your DukeCard)



- **Cryptographic token (AKA one-time passwords)**
  - Holds crypto key that can't (easily) be extracted
  - Uses it to generate a time-sync'd key stream

Types of tokens (2)

- **Communication device** (i.e., your phone)
  - Relies on real-time and secure communication
  - Good: Dedicated app with cryptographic secrets (e.g. Duo)
  - Bad: Using SMS (text messaging)
    - Many examples of SMS hijacking:
      Every helpdesk employee at your mobile provider can do it (either because they were fooled or they're evil)!
    - Better than nothing, though…

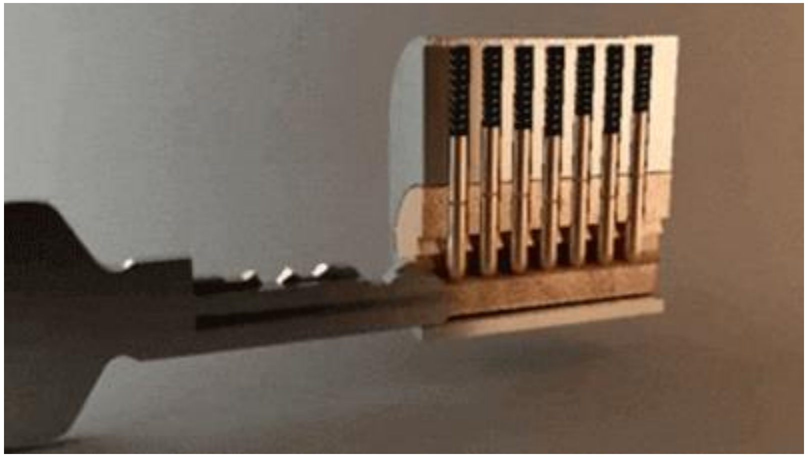


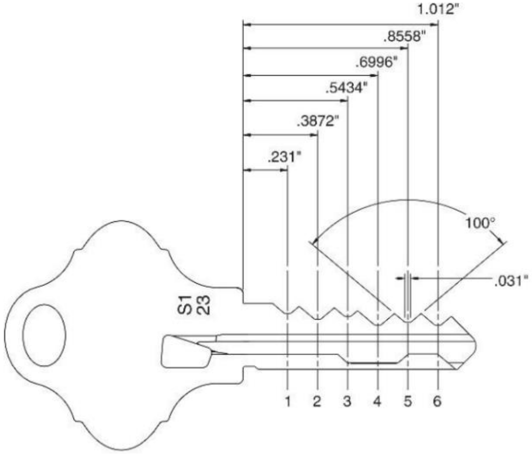
- **Authentication token**
  - Similar to cryptographic token from before, but communicates digitally rather than with displayed one-time passwords
  - The "cool" version of multi-factor authentication

A common model made by Yubikey

Types of tokens (3)

- **Physical keys** (they're made of metal and you have some)
  - Many different types, same idea: mechanically unbind a lock





MACS = 7

Increment:	0.15"
Progression:	Two Step
Blade Width:	.343"
Depth Tolerance:	+ .002" - 0
Spacing Tolerance:	± .001"

Root Depths

0	.335"
1	.320"
2	.305"
3	.290"
4	.275"
5	.260"
6	.245"
7	.230"
8	.215"
9	.200"

  - Turns out you can attack physical locks many different ways (covered later when we get to physical security)

- **Fallback passwords**
  - Long, random single use passwords that are written down or stored
  - Kept in a secure location for exception situations (e.g. in response to an account hijack)

More on contactless communication

- Recall: smart cards may be **contactless**
    - Has CPU, memory, ROM, maybe even non-volatile storage (EEPROM/flash)
- Terminology and standards:
    - **RFID**: Radio Frequency Identification
        - Broad category
        - Usually *powered* wirelessly (inductively or via RF pulse)
        - May be very short range (like DukeCard) or longer (Duke parking pass)
        - May be very dumb (“just transmit this string”) or more advanced (“execute this encrypted read/write command”)
    - **NFC**: Near Field Communication
        - A collection of standards for two-way communication based on RFID
        - Generally on the smarter side in terms of protocol
        - Supported by modern mobile phones
            - Powers things like “ApplePay”, “GooglePay”, etc.
            - Your DukeCard is NFC,
              and your phone can act as a DukeCard using NFC

Four means of authentication

Something you **know**	• Password, PIN, answers to prearranged questions
Something you **have** (*token*)	• Smartcard, electronic keycard, physical key
Something you **are** (*static biometrics*)	• Fingerprint, retina, face
Something you **do** (*dynamic biometrics*)	• Voice pattern, handwriting

Biometric basics

- Authenticate based on unique physical characteristics (pattern recognition)
- More complex/expensive than previous techniques
- Common characteristics:
  - Fingerprint
  - Face
- Less common:
  - Hand geometry
  - Retinal pattern
  - Iris
  - Signature
  - Voice

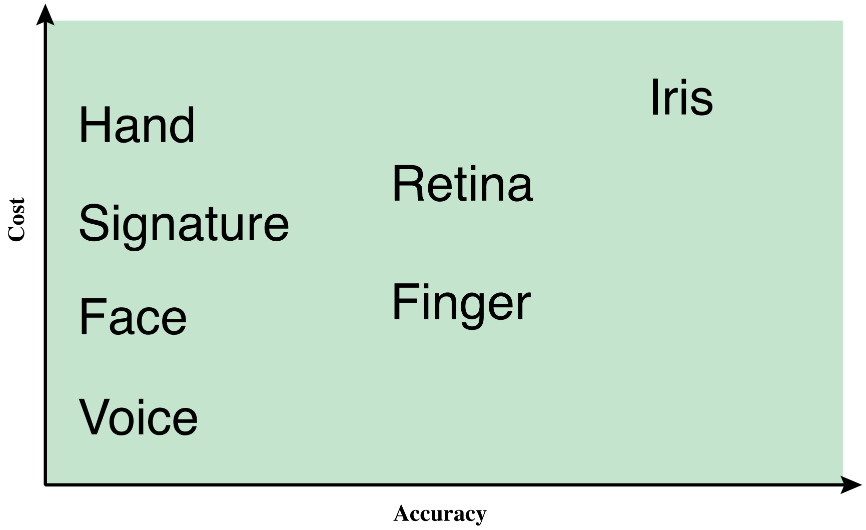
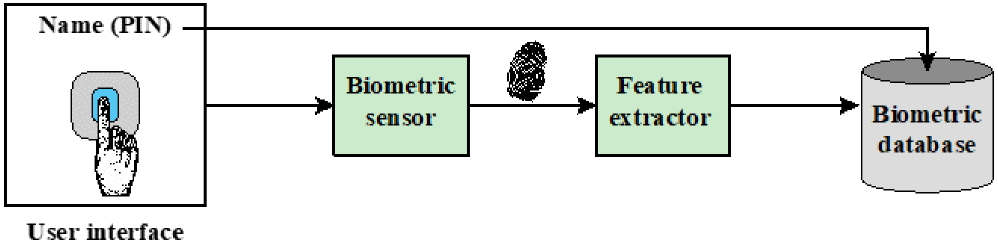


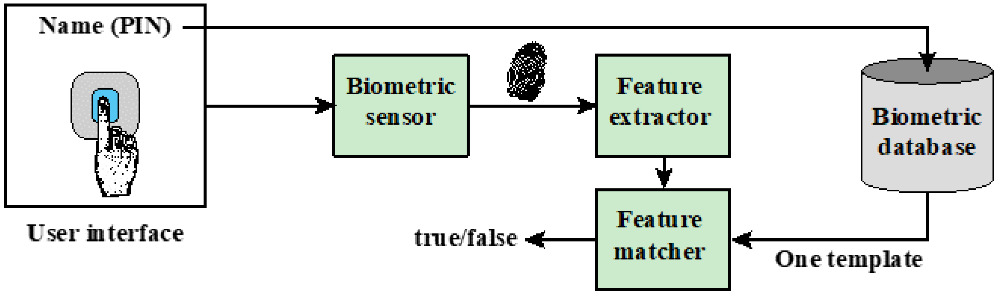
Figure 3.8 Cost Versus Accuracy of Various Biometric Characteristics in User Authentication Schemes.

Processes of biometric authentication



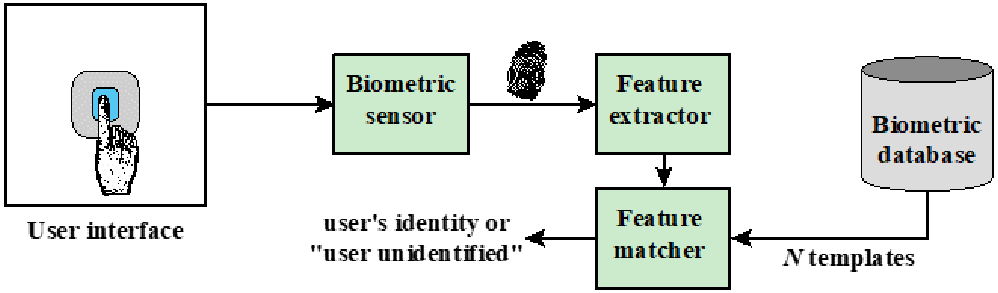
(a) Enrollment

- **Enrollment**: Add new people



(b) Verification

- **Verification**: User asserts identity and proves it

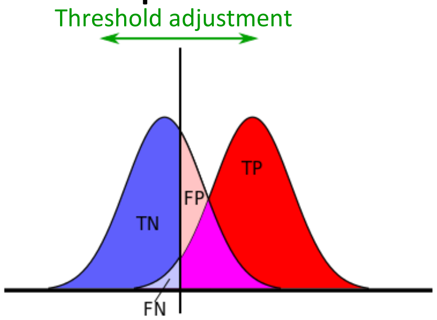


(c) Identification

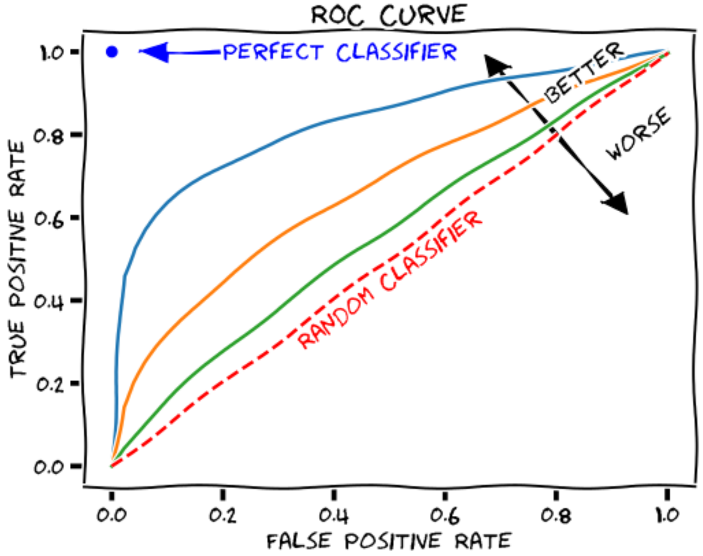
- **Identification**: Pick out which user the given biometric corresponds to (harder)

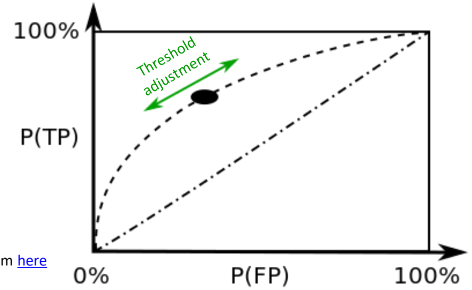
Analyzing biometric accuracy

- Biometric is pattern matching; naturally imprecise (probabilistic)
  - Will get a **match score**, system *accepts* when score ≥ **threshold**
- Metrics to evaluate a biometric system:
  - **False Accept Rate (FAR)**: Probability it allows the wrong person
    = False positive (FP) rate
  - **False Reject Rate (FRR)**: Probability it disallows the right person
    = False negative (FN) rate
  - **Receiver Operating Characteristic (ROC)**: Comparison of the FAR+FRR with respect to threshold (a general concept for *any* classifier)



TP	FP
FN	TN





Figures from here

Remote authentication

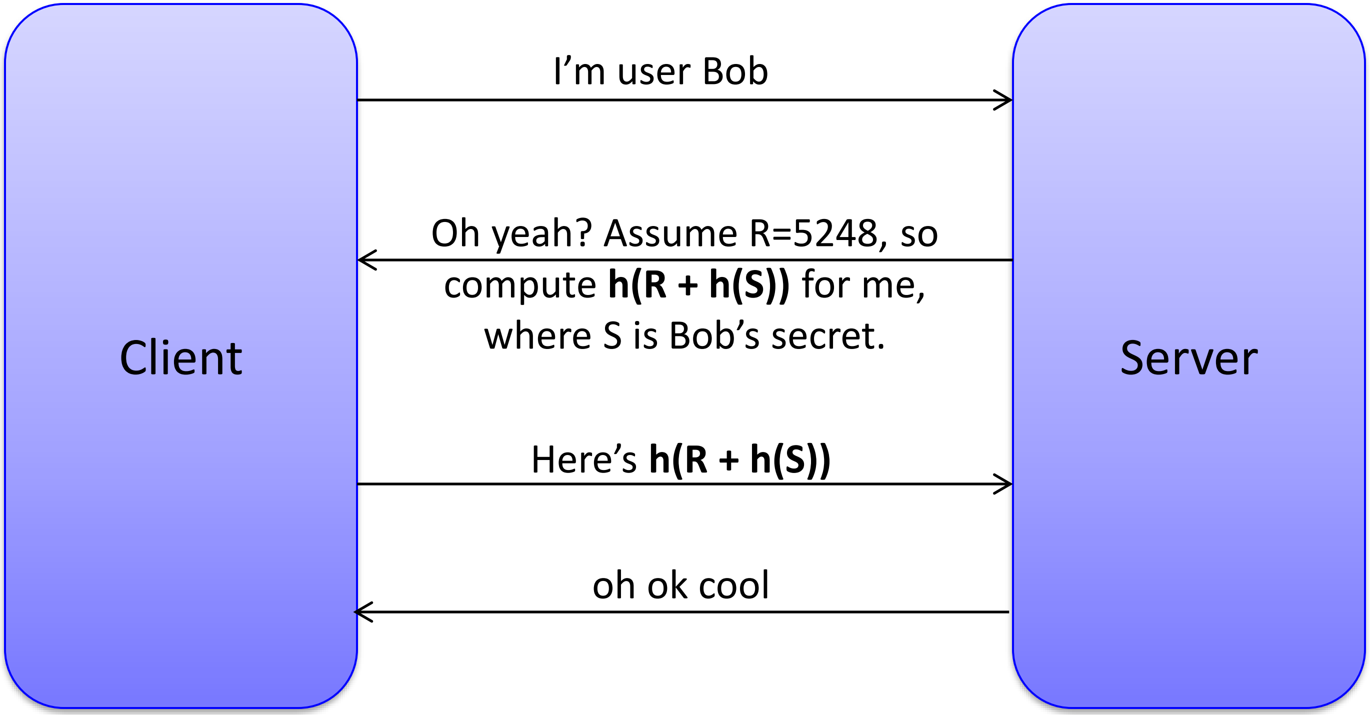
What about the network?

- Authentication over a network is more complex – more to worry about
  - **Eavesdropping**:
    - Capturing a credential (allowing attacker to login)
    - Capturing a session cookie (evidence of authentication, allows attacker to act as user)
  - **Replay attacks**: Even if attacker doesn’t know credential, can they blindly replay the packets to login?
    - Example: a “pass the hash” attack

- Solution: Various **challenge-response** schemes

A basic challenge-response scheme

- Assume we have some authentication secret S
    - Password, 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))**

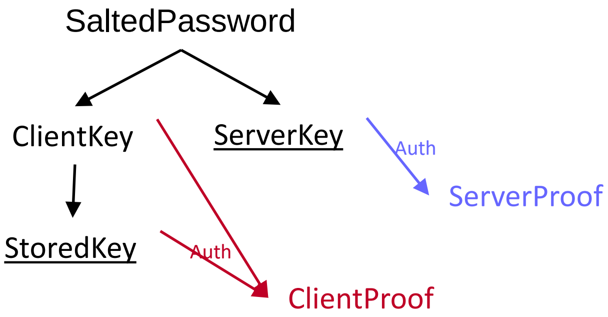
oh ok cool

Server

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**)

Mutations done to the salted password



SaltedPassword

ClientKey

ServerKey

Auth

ServerProof

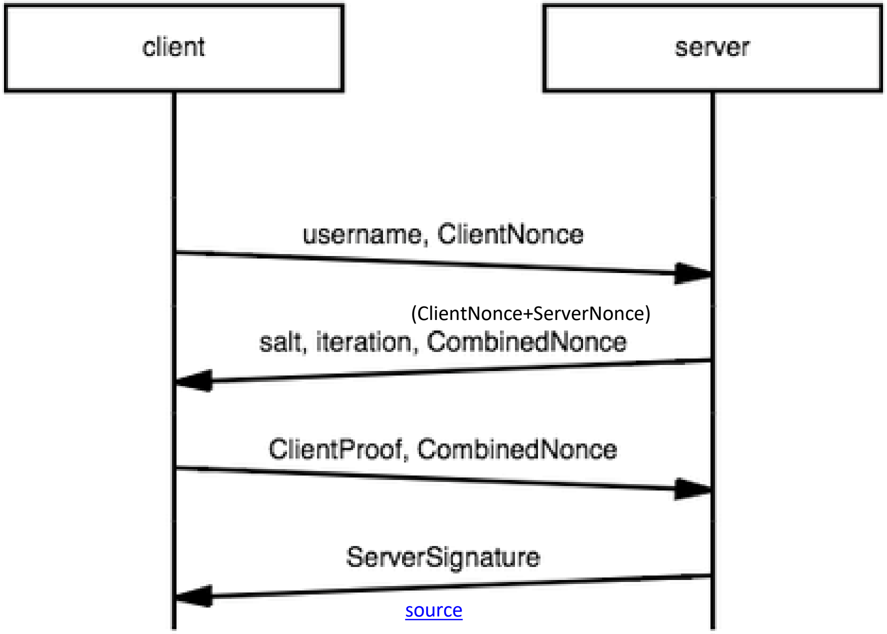
StoredKey

Auth

ClientProof

Black = computed by server when account is created
Underline = stored by server
Red = computed by client during auth
Blue = computed by server during auth

Communications sequence



client

server

username, ClientNonce

(ClientNonce+ServerNonce)
salt, iteration, CombinedNonce

ClientProof, CombinedNonce

ServerSignature

source

SaltedPassword = {salted hash of password}
ClientKey = HMAC(SaltedPassword, "Client Key")
StoredKey = H(ClientKey)
ServerKey = HMAC(SaltedPassword, "Server Key")

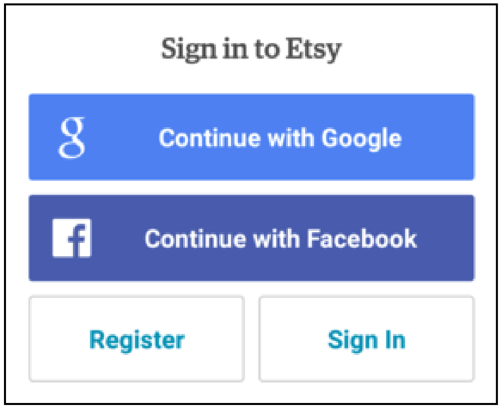
Auth = {username, salt, iteration, CombinedNonce}

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

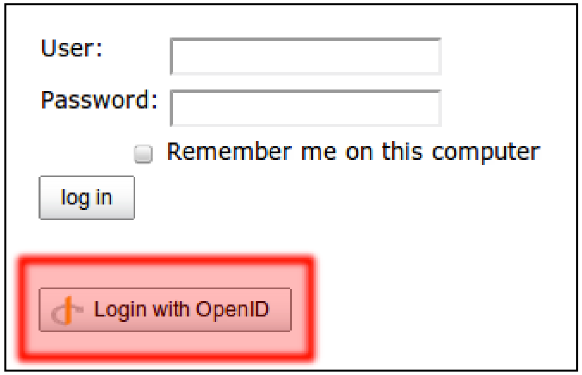
For more, see Wikipedia or this article

Identity Federation

- **Identity Federation**: System to allow an organization to trust identities/credentials managed by another organization
  - Allows you to provide access to users from external orgs (and vice versa)
- 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

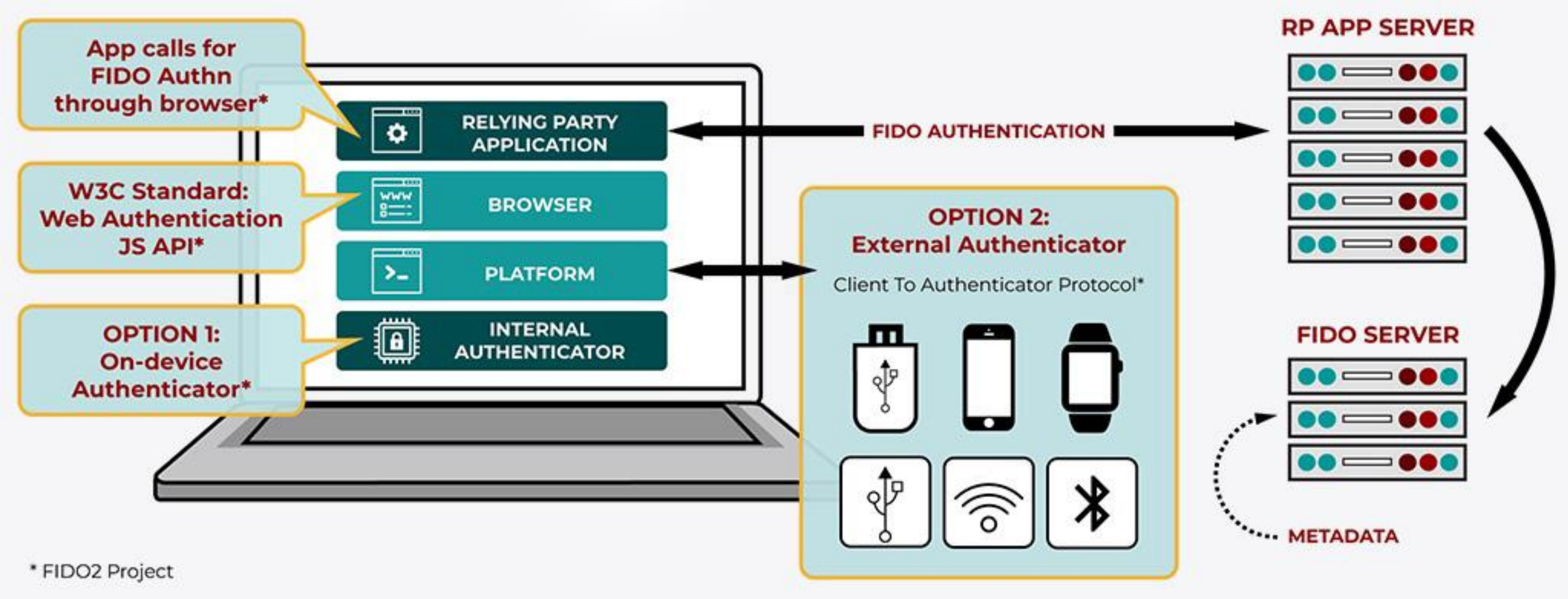
- Want a standard to support federation from any provider? **OAuth**
- Duke has an authentication system: **Duke NetID**
  - You can write apps that use OAuth to allow login via Duke NetID!

Multifactor Authentication (MFA)

Multifactor authentication (MFA)

- Now that we've covered the modes of authentication (something you **know**/**have**/**are**/**do**), definition is easy:
  - ***Multifactor Authentication****: Require more than one of those categories.* (that's all)
- In *practice*, today it usually means password + token.
  - Lame: Password + SMS
  - Better: Password + actual token or app
- Looking forward:
  - Trusted Platform Modules (TPMs) are hardware chips that can securely hold cryptographic secrets without leaking them (unless there's a flaw…)
  - Modern standard: **WebAuthn** – use TPM to make MFA easy

WebAuthn: Practical MFA of the future


App calls for FIDO Authn through browser*
W3C Standard: Web Authentication JS API*
OPTION 1: On-device Authenticator*
RELYING PARTY APPLICATION
BROWSER
PLATFORM
INTERNAL AUTHENTICATOR
FIDO AUTHENTICATION
RP APP SERVER
OPTION 2: External Authenticator
Client To Authenticator Protocol*
FIDO SERVER
METADATA
* FIDO2 Project

- WebAuthn incorporates FIDO authentication (an open standard)
  - Web app: Implements WebAuthn standard to ask for a login
  - Browser: Needs WebAuthn support, hooks into support from OS
  - OS: Provides a Client-To-Authenticator Protocol (CTAP). May use:
    - Internal authenticator (using TPM chip), or
    - External token (phone, watch, USB security token)

These store cryptographic keys, never divulge them, give proof via signature

Figure from https://fidoalliance.org/fido2/

Access control

So you’ve proven who you are, but what are you allowed to do?

Topics

- Core concepts
- Access control policies:
  - DAC
    - UNIX file system
  - MAC
  - RBAC

Subjects, Objects, Actions, and Rights

Subject
(initiator)

- The thing making the request (e.g. the user)

Verb
(request)

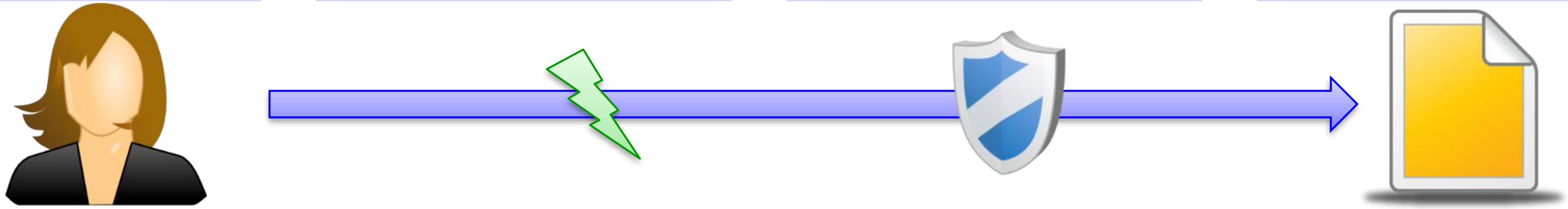
- The operation to perform (e.g., read, delete, etc.)

Right
(permission)

- A specific ability for the subject to do the action to the object.

Object
(target)

- The thing that's being hit by the request (e.g., a file).



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
  - *We're gonna skip this, since I've never seen anyone care about it IRL*

Topics

- Core concepts
- Access control policies:
  - DAC
    - UNIX file system
  - MAC
  - RBAC

Discretionary Access Control (DAC)

- **Discretionary Access Control (DAC)**: Scheme in which an entity may enable another entity to access some resource
  - Often provided using an access matrix: *subjects* × *objects*
  - Each entry shows the access rights of that subject to that object

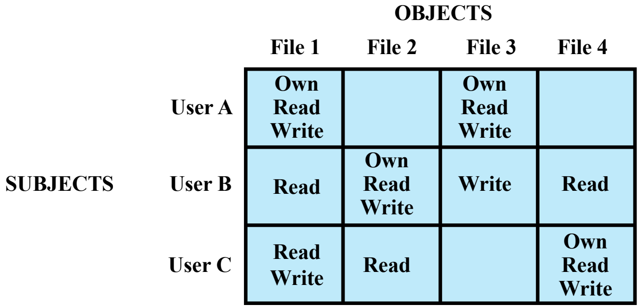
Pseudocode

```
bool IsActionAllowed(subject, object, action) {
  if (action $\in$ get_permissions(subject,object))
    return true
}
```

Implementation

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

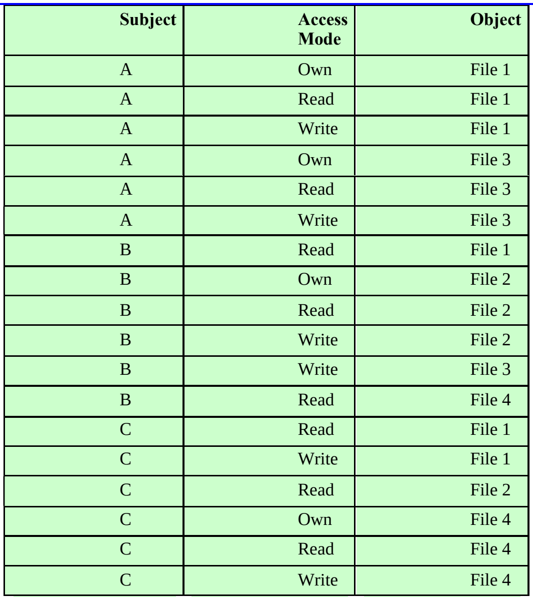
Matrix



		OBJECTS			
		File 1	File 2	File 3	File 4
SUBJECTS	User A	Own Read Write		Own Read Write	
	User B	Read	Own Read Write	Write	Read
	User C	Read Write	Read		Own Read Write

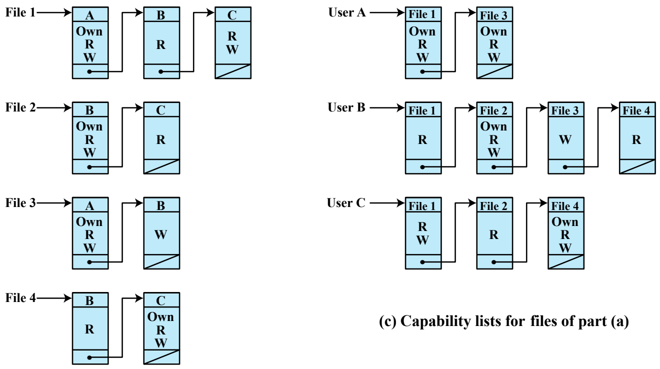
(a) Access matrix

Flat list



Subject	Access Mode	Object
A	Own	File 1
A	Read	File 1
A	Write	File 1
A	Own	File 3
A	Read	File 3
A	Write	File 3
B	Read	File 1
B	Own	File 2
B	Read	File 2
B	Write	File 2
B	Write	File 3
B	Read	File 4
C	Read	File 1
C	Write	File 1
C	Read	File 2
C	Own	File 4
C	Read	File 4
C	Write	File 4

Linked list



(c) Capability lists for files of part (a)

(b) Access control lists for files of part (a)

Figure 4.2 Example of Access Control Structures

UNIX Philosophy

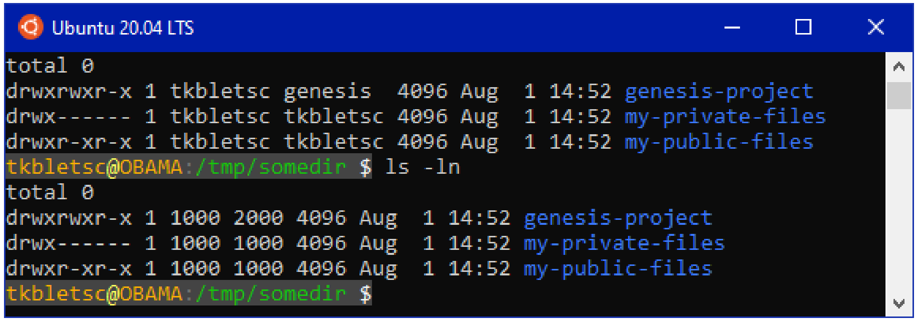
- "UNIX" here includes Linux, MacOSX, and traditional UNIX
- Major tenet of UNIX philosophy: ***everything is a file***
  - Why?
  - **Flexibility**: If you build an API to access files, you can use it for everything ☺
  - **Security**: If you build a permission system for files, you can use it for everything ☺
- How everything is a file:
  - Hardware devices show up as files under **`/dev`**
  - Info and controls for the running kernel are simulated in **`/proc`** and **`/sys`**
  - You can attach ("**mount**") storage devices to directories all under one global hierarchy
  - You can even turn a pipe or socket into a named file!

UNIX File Access Control

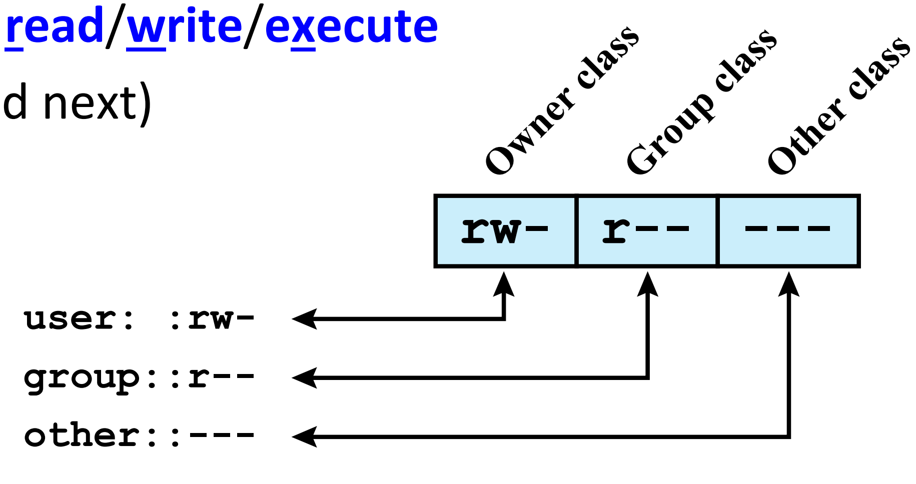
- Your **disk** is a dumb flat array of blocks that can be read/written
- A **filesystem** organizes this, handles allocation of disk regions to **files**, lets you organize these files into hierarchical **directories**.
- (Most) UNIX filesystems store file metadata in **inodes** (index nodes)
  - Inodes store metadata about a file/directory, including ownership/permissions
  - They live on disk in an inode table; in memory in a kernel inode cache
- **Directories** are special files that list names + inode numbers
- There are a few other special file types:
  - **Symbolic links** (also known as **symlinks** or **soft links**)
  - **Device files** (**character** or **block**)
  - **Named pipes** (also known as **fifos**)
  - **Named sockets** (like two-way fifos)

UNIX File Access Control Basics

- **Users** have numbers called User ID numbers (“**uid**”)
- Users can belong to one or more **groups**;
  groups have numbers called Group ID numbers (“**gid**”)
- A file is *owned* by a user (**uid**) and a group (**gid**)
  - The reference is numeric; `ls` translates numbers to names for you; can turn off with `-n`
- Twelve permission bits applied to file (file “**mode**”)
  - Lower 9 bits: **user**/**group**/**others** : **read**/**write**/**execute**
  - Upper 3 bits: “Weird” ones (covered next)



```
total 0
drwxrwxr-x 1 tkbletsc genesis  4096 Aug  1 14:52 genesis-project
drwx------ 1 tkbletsc tkbletsc 4096 Aug  1 14:52 my-private-files
drwxr-xr-x 1 tkbletsc tkbletsc 4096 Aug  1 14:52 my-public-files
tkbletsc@OBAMA:/tmp/somedir $ ls -ln
total 0
drwxrwxr-x 1 1000 2000 4096 Aug  1 14:52 genesis-project
drwx------ 1 1000 1000 4096 Aug  1 14:52 my-private-files
drwxr-xr-x 1 1000 1000 4096 Aug  1 14:52 my-public-files
tkbletsc@OBAMA:/tmp/somedir $
```



(a) Traditional UNIX approach (minimal access control list)

UNIX File Access Control Basics

- Change a file’s owner with **`chown`** (changes uid)
- Change a file’s group with **`chgrp`** (changes gid)
- Change a file’s mode (permissions) with **`chmod`** (changes mode bits)
  - Can express in base-8 octal: **`chmod 750`** yields **`rwxr-x---`**
  - Can express symbolically: **`chmod u+rw`** turns on owner’s read/write

    user/group/others read/write/execute
- The other three bits:
  - **SetUID** (`u+s`) and **SetGID** (`g+s`):
    - Executables run that have this bit run as the user/group that owns it
    - A way to allow privilege escalation, either
      legitimately, like for **`sudo`**, or
      illegitimately, as in a backdoor created by attackers
  - **Sticky bit** (`+t`):
    - Applied to directories; when set, only the owner of any file in the directory can rename, move, or delete that file – used for e.g. **`/tmp`**
- The **root** user (uid 0) is immune from permission bit limitations.
  - Hence using `sudo` to carry out chown/chgrp/chmod commands when you otherwise couldn’t.

Sidebar: Hard vs soft links

- Directories are special files that list file names and inode numbers
- **Hard link**: When multiple directory entries refer to the same **inode**
  - Such “files” are actually the same content; change one = change all
  - Useful for creating cheap “clones” of files, no extra storage
- **Soft link**: A special file that refers to another **path**
  - Also called **symbolic link** or **symlink**.
  - Path can be relative or absolute
  - Can traverse file systems or even point to nonexistent things
  - Can be used as file system organization “duct tape”
    - Example: Symlink a long, complex path to a simpler place, e.g.:

```
$ ln -s /remote/codebase/projectX/beta/current/build ~/mybuild
$ cd ~/mybuild
```

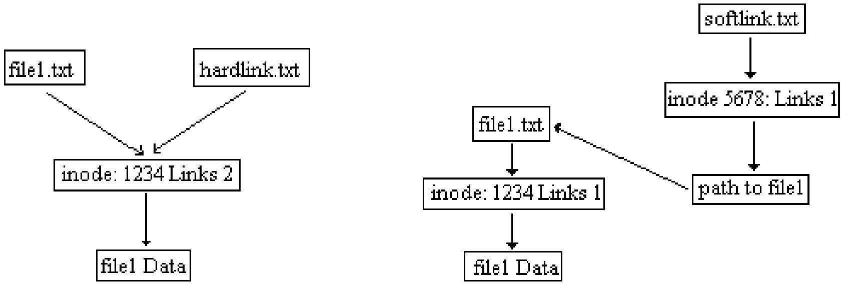
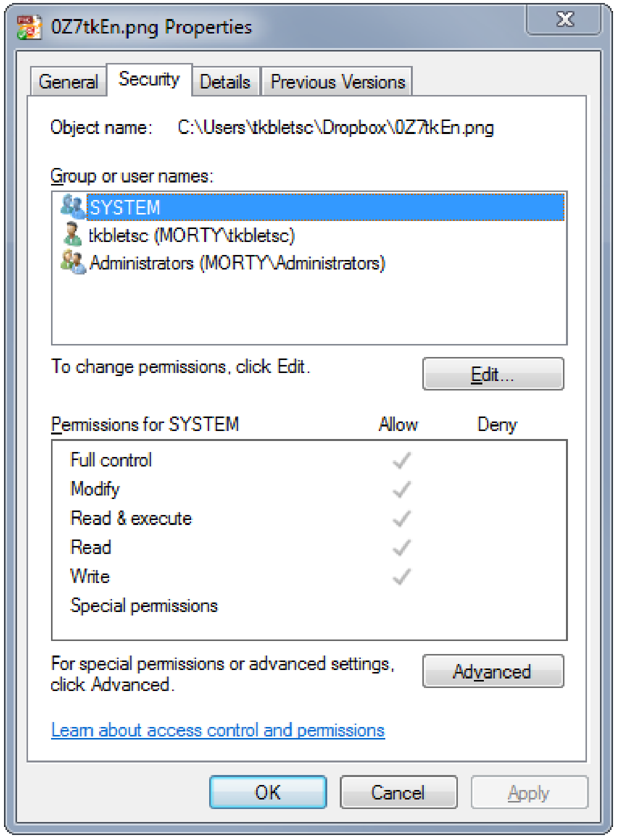


Figure from “Computer Forensics: Unix File Systems” by Thomas Schwarz (Santa Clara Univ)

File system access control lists (ACLs)

- Issue: UNIX model can’t represent all permission situations (e.g. multiple groups or users having access); use **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



0Z7tkEn.png Properties

Object name: C:\Users\tkbletsc\Dropbox\0Z7tkEn.png

Group or user names:

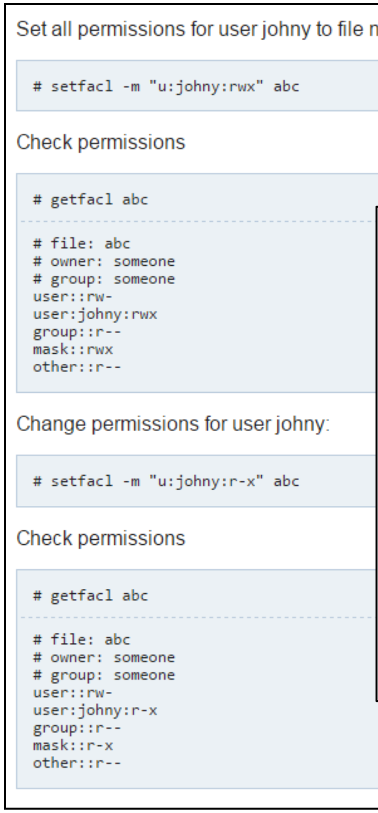
- SYSTEM
- tkbletsc (MORTY\tkbletsc)
- Administrators (MORTY\Administrators)

To change permissions, click Edit.

Permissions for SYSTEM	Allow	Deny
Full control	✓	
Modify	✓	
Read & execute	✓	
Read	✓	
Write	✓	
Special permissions		

For special permissions or advanced settings, click Advanced.

Examples of Linux ACL commands



Set all permissions for user johny to file named "abc":

```
# setfacl -m "u:johny:rwx" abc
```

Check permissions

```
# getfacl abc

# file: abc
# owner: someone
# group: someone
user::rw-
user:johny:rwx
group::r--
mask::rwx
other::r--
```

Change permissions for user johny:

```
# setfacl -m "u:johny:r-x" abc
```

Check permissions

```
# getfacl abc

# file: abc
# owner: someone
# group: someone
user::rw-
user:johny:r-x
group::r--
mask::r-x
other::r--
```

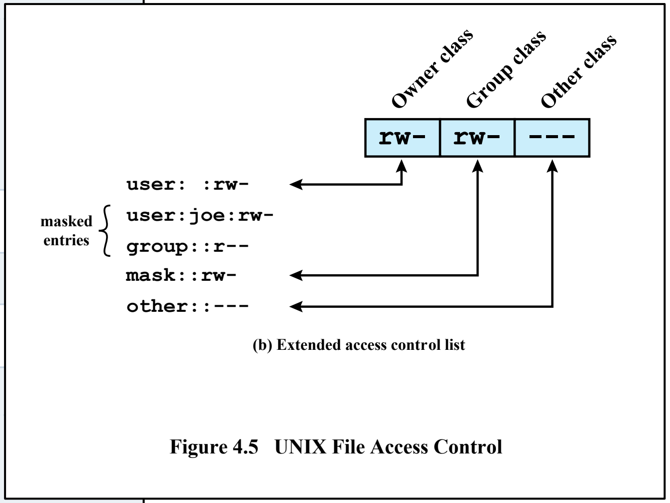


Figure 4.5 UNIX File Access Control

From Arch Wiki

Topics

- Core concepts
- Access control policies:
  - DAC
    - UNIX file system
  - MAC
  - RBAC

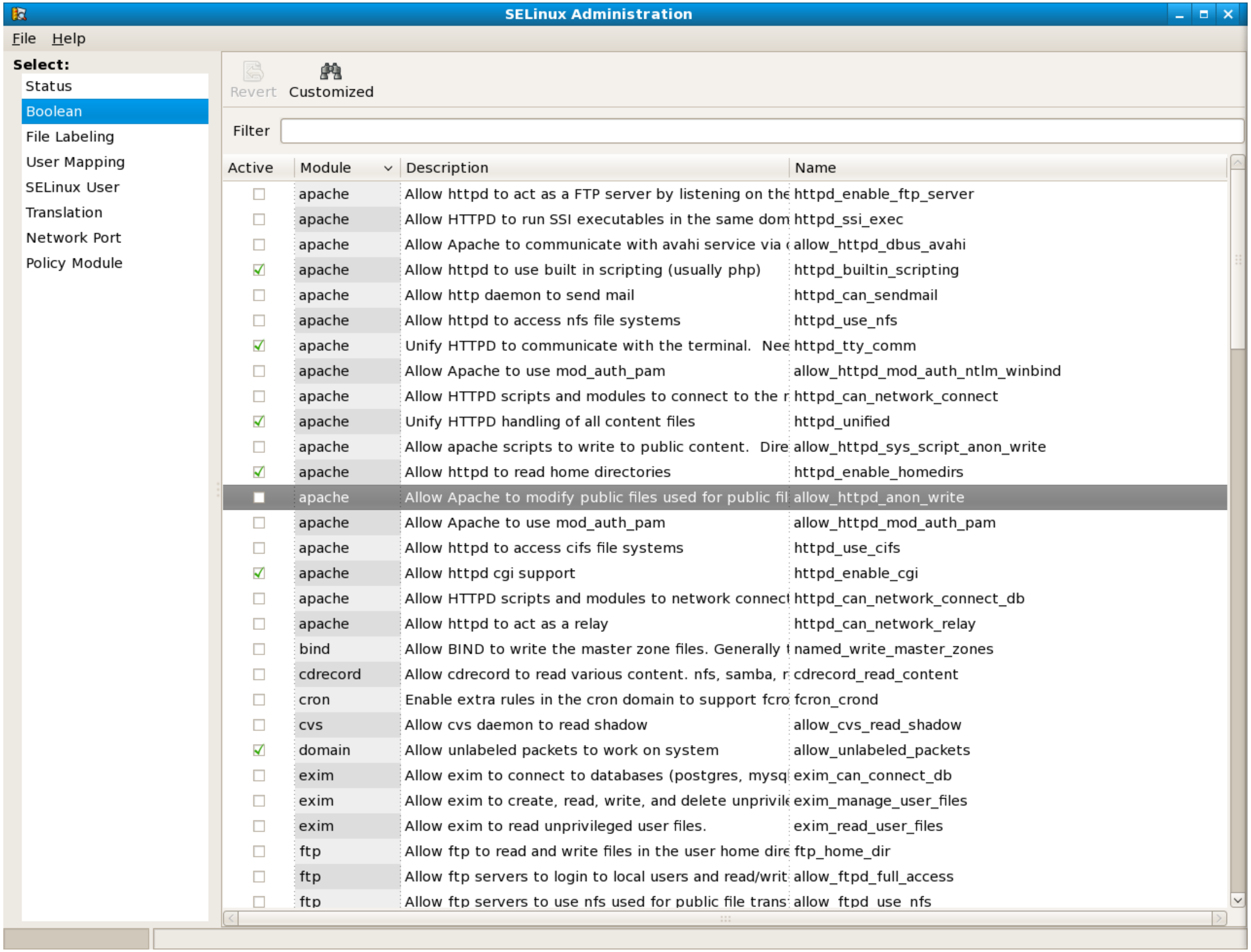
MAC example: SELinux

- 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

Pseudocode

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

MAC example: SELinux



Active	Module	Description	Name
☐	apache	Allow httpd to act as a FTP server by listening on the	httpd_enable_ftp_server
☐	apache	Allow HTTPD to run SSI executables in the same dom	httpd_ssi_exec
☐	apache	Allow Apache to communicate with avahi service via (	allow_httpd_dbus_avahi
☑	apache	Allow httpd to use built in scripting (usually php)	httpd_builtin_scripting
☐	apache	Allow http daemon to send mail	httpd_can_sendmail
☐	apache	Allow httpd to access nfs file systems	httpd_use_nfs
☑	apache	Unify HTTPD to communicate with the terminal. Nee	httpd_tty_comm
☐	apache	Allow Apache to use mod_auth_pam	allow_httpd_mod_auth_ntlm_winbind
☐	apache	Allow HTTPD scripts and modules to connect to the r	httpd_can_network_connect
☑	apache	Unify HTTPD handling of all content files	httpd_unified
☐	apache	Allow apache scripts to write to public content. Dire	allow_httpd_sys_script_anon_write
☑	apache	Allow httpd to read home directories	httpd_enable_homedirs
☐	apache	Allow Apache to modify public files used for public fil	allow_httpd_anon_write
☐	apache	Allow Apache to use mod_auth_pam	allow_httpd_mod_auth_pam
☐	apache	Allow httpd to access cifs file systems	httpd_use_cifs
☑	apache	Allow httpd cgi support	httpd_enable_cgi
☐	apache	Allow HTTPD scripts and modules to network connect	httpd_can_network_connect_db
☐	apache	Allow httpd to act as a relay	httpd_can_network_relay
☐	bind	Allow BIND to write the master zone files. Generally t	named_write_master_zones
☐	cdrecord	Allow cdrecord to read various content. nfs, samba, r	cdrecord_read_content
☐	cron	Enable extra rules in the cron domain to support fcro	fcron_crond
☐	cvs	Allow cvs daemon to read shadow	allow_cvs_read_shadow
☑	domain	Allow unlabeled packets to work on system	allow_unlabeled_packets
☐	exim	Allow exim to connect to databases (postgres, mysq	exim_can_connect_db
☐	exim	Allow exim to create, read, write, and delete unprivile	exim_manage_user_files
☐	exim	Allow exim to read unprivileged user files.	exim_read_user_files
☐	ftp	Allow ftp to read and write files in the user home dire	ftp_home_dir
☐	ftp	Allow ftp servers to login to local users and read/writ	allow_ftpd_full_access
☐	ftp	Allow ftp servers to use nfs used for public file trans	allow_ftpd_use_nfs

Topics

- Core concepts
- Access control policies:
  - DAC
    - UNIX file system
  - MAC
  - RBAC

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. ☹

RBAC

- 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

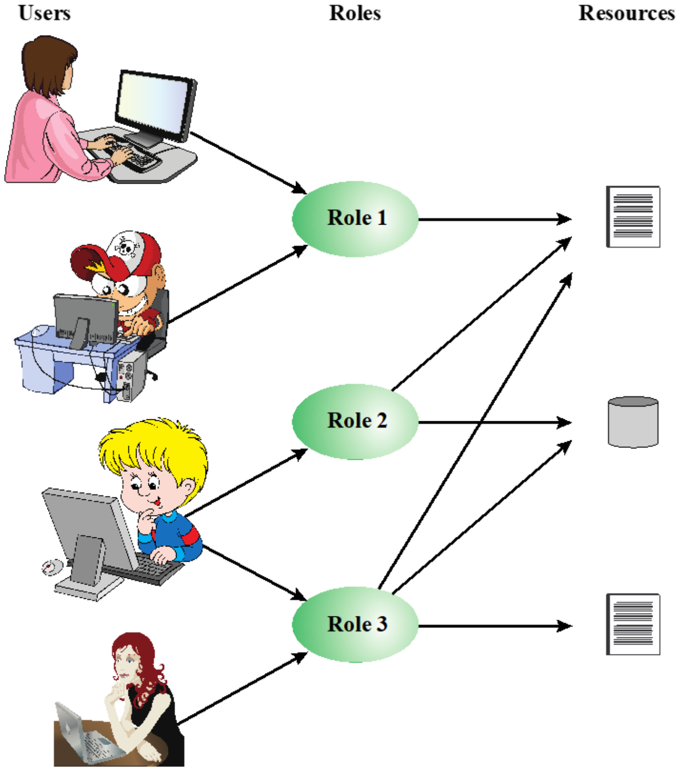
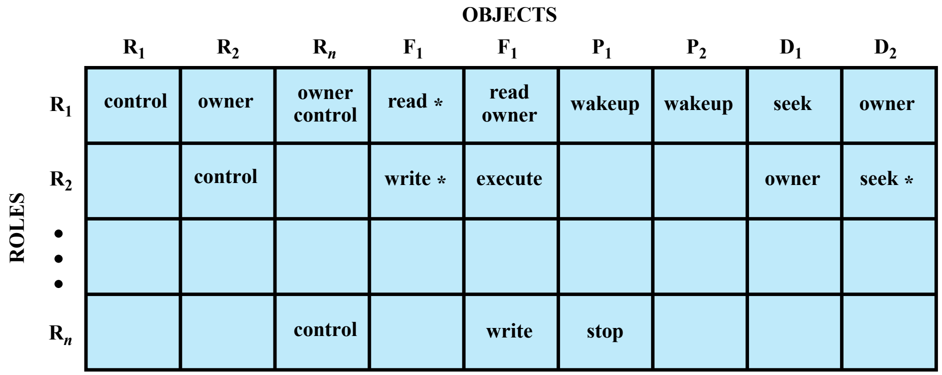


Figure 4.6 Users, Roles, and Resources

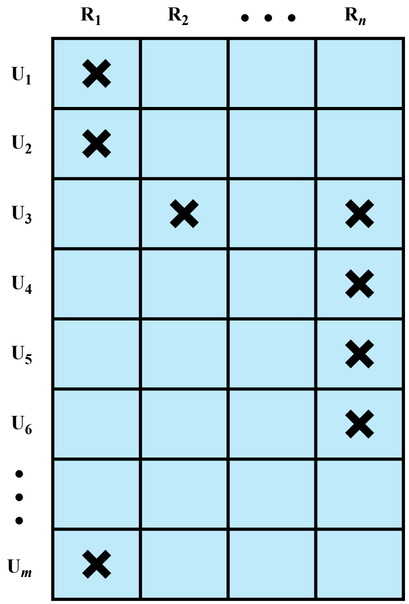
RBAC implementation

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



ROLES	OBJECTS								
	R_1	R_2	R_n	F_1	F_1	P_1	P_2	D_1	D_2
R_1	control	owner	owner control	read *	read owner	wakeup	wakeup	seek	owner
R_2		control		write *	execute			owner	seek *
⋮									
R_n			control		write	stop			

Figure 4.7 Access Control Matrix Representation of RBAC



	R_1	R_2	• • •	R_n
U_1	✕			
U_2	✕			
U_3		✕		✕
U_4				✕
U_5				✕
U_6				✕
⋮				
U_m	✕			

Pseudocode

```
bool IsActionAllowed(subject, object, action) {
  if (action ∈ get_permissions(subject.role,object))
    return true
}
```

Any questions?