Lecture #4: OS Security Concepts

Administrivia

- Project 1 is out now
  - Start now: Don't wait for the last minute

Access Control

- Some resources (files, web pages, …) are sensitive.
- How do we limit who can access them?
- This is called the ***access control*** problem
- A ***foundational*** problem when building a secure system:
  - We ***must*** be able to specify who is allowed and who is forbidden from accessing something
  - We ***must*** be able to enforce our specification

Access Control Fundamentals

- ***Subject*** = a user, process, …
  (something who is accessing resources)
- ***Object*** = a file, device, web page, …
  (a resource that can be accessed)
- ***Policy*** = the restrictions we'll enforce
- ***Mechanism*** = what enforces the policy
- access(S, O) = ***true***
  if subject S is allowed to access object O
- access(S, O) = ***false***
  if subject S is forbidden to access object O
- Defaults matter:
  - If unspecified, is the default "true" (default-allow) or "false" (default-deny)

Example

- access(Alice, Alice’s Facebook wall) = true
- access(Alice, Bob’s Facebook wall) = true
- access(Alice, Charlie’s Facebook wall) = false
- access(Friend(Alice), Alice’s Facebook wall) = true
  - Reasoning in terms of “groups” can often make the logic easier

- access(nweaver, /home/cs161/gradebook) = true
- access(Alice, /home/cs161/gradebook) = false
  - alert(Alice, attempt to access /home/cs161/gradebook) = hell yah

Access Control Matrix

- access(S, O) = true
  if subject S is allowed to access object O

	Alice's wall	Bob's wall	Charlie's wall	…
Alice	true	true	false	
Bob	false	true	false	
…				

Permissions

- We can have finer-grained permissions,
  e.g., read, write, execute.

- access(daw, /cs161/grades/alice) = {read, write}
  access(alice, /cs161/grades/alice) = {read}
  access(bob, /cs161/grades/alice) = {}

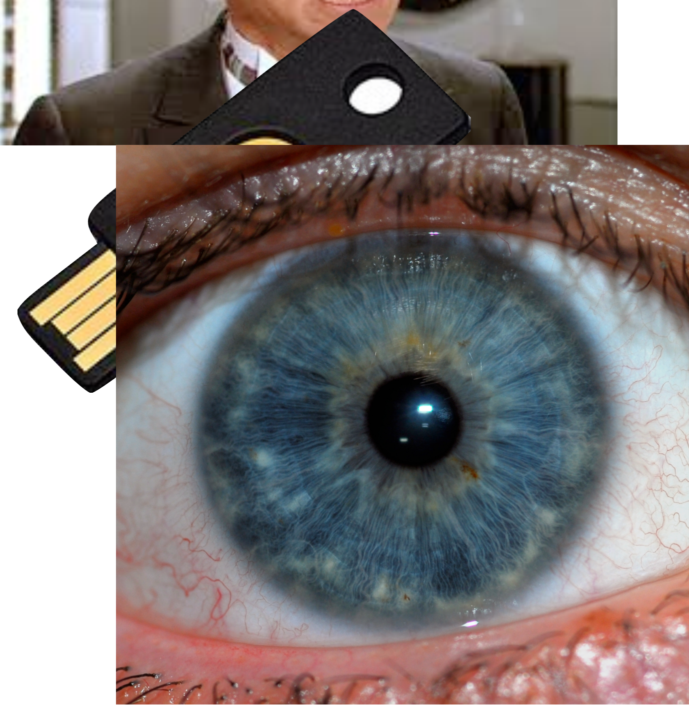
	/cs161/grades/alice
nweaver	read, write
alice	read
bob	-

Access Control

- ***Authorization***: who should be able to perform which actions
  - Nick, Reluca, and the TAs are the only ones ***authorized*** to access the grade database
- ***Authentication***: verifying who is requesting the action
  - Yes, this is Nick accessing the grade database
- ***Audit***: a log of all actions, attributed to a particular principal
  - Nick gave John Smith an A+
- ***Accountability***: hold people legally responsible for actions they take
  - John Smith hijacked Nick's credentials and now his grade is an F

Establishing *Identity*

- In order to enforce access control the system needs to know who is whom…
- "Something you know"
  - Almost certainly a password
- "Something you have"
  - Security token, cellphone, etc
- "Something you are"
  - Fingerprint, iris scan, etc



Two Factor Verification

- Assumption: An attacker can easily grab one factor
  - Guess/determine your password
  - Steal your keys
  - Clone a fingerprint ("Gummy fingers")
- But it is ***much*** harder for an attacker to grab ***two*** factors
  - But they have to be independent:
    If both "factors" are something you know, its not two-factor!
- Two-factor can often serve to detect attacks
  - EG, SMS notification on login
- Good 2-factor prevents, not just mitigates attacks
  - FiDO U2F:
    The second factor is bound to the site:
    A phishing link ***can not*** use the second factor
  - If you exclusively use Crome as your web browser, buy yourself a Fido U2F token!

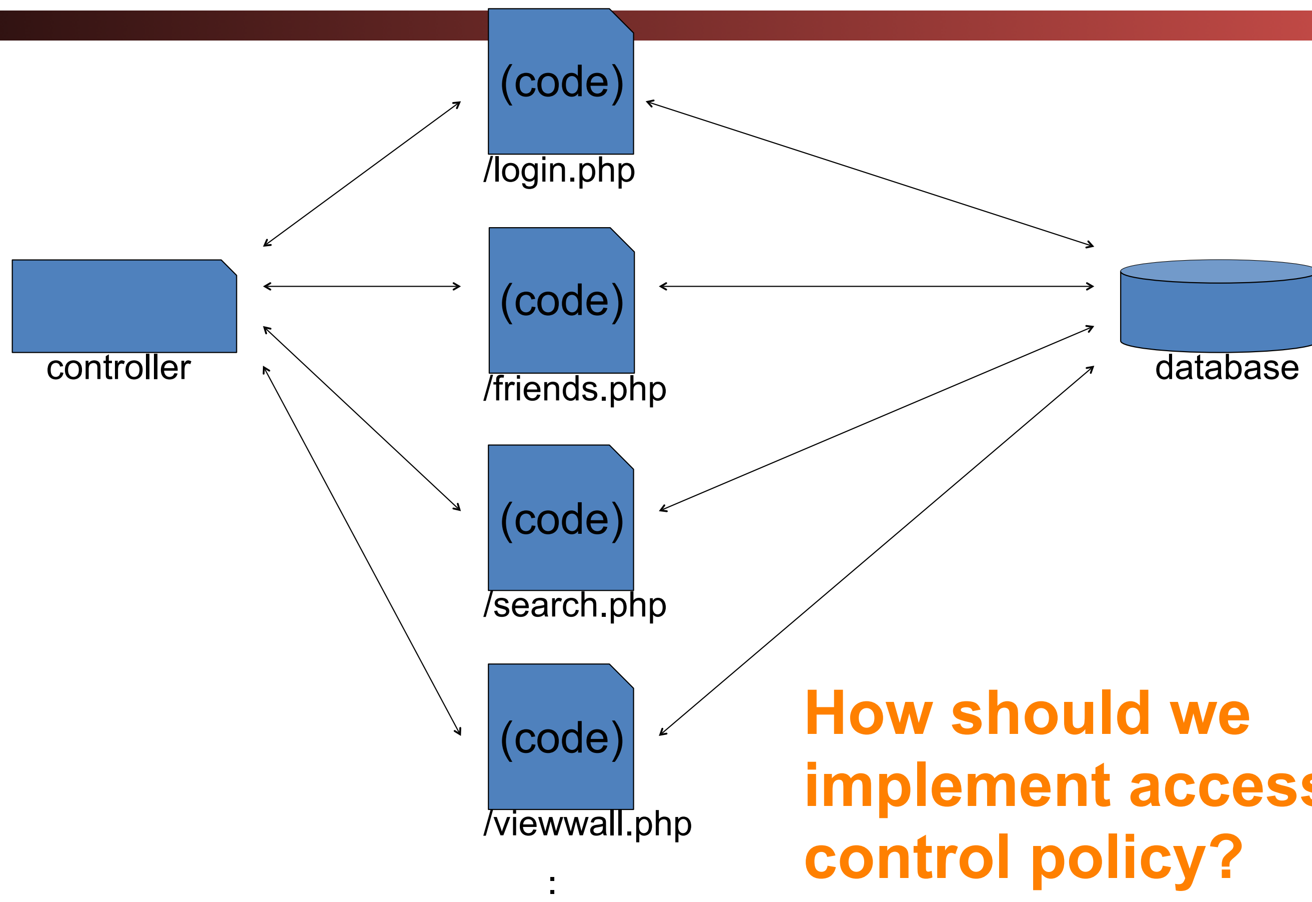
Recovery Mechanisms

- Unfortunately people aren't perfect
  - They forget passwords, lose authentication tokens, and even suffer accidental amputation...
- At scale it gets worse:
  - If you have 10M users, you're going to have people losing passwords ***all the time***
- So recovery proves to be the weakness:
  - Password recovery channels: email, SMS, etc
    - But what happens with a lost phone?
  - "Knowledge Based Authentication": stuff about your finances etc... That the black market knows
- Practical upshot:
  - Lock down the keystone recovery mechanisms:
    Make sure your phone requires ID in person to change
    Make sure your master email is well secured

Web security

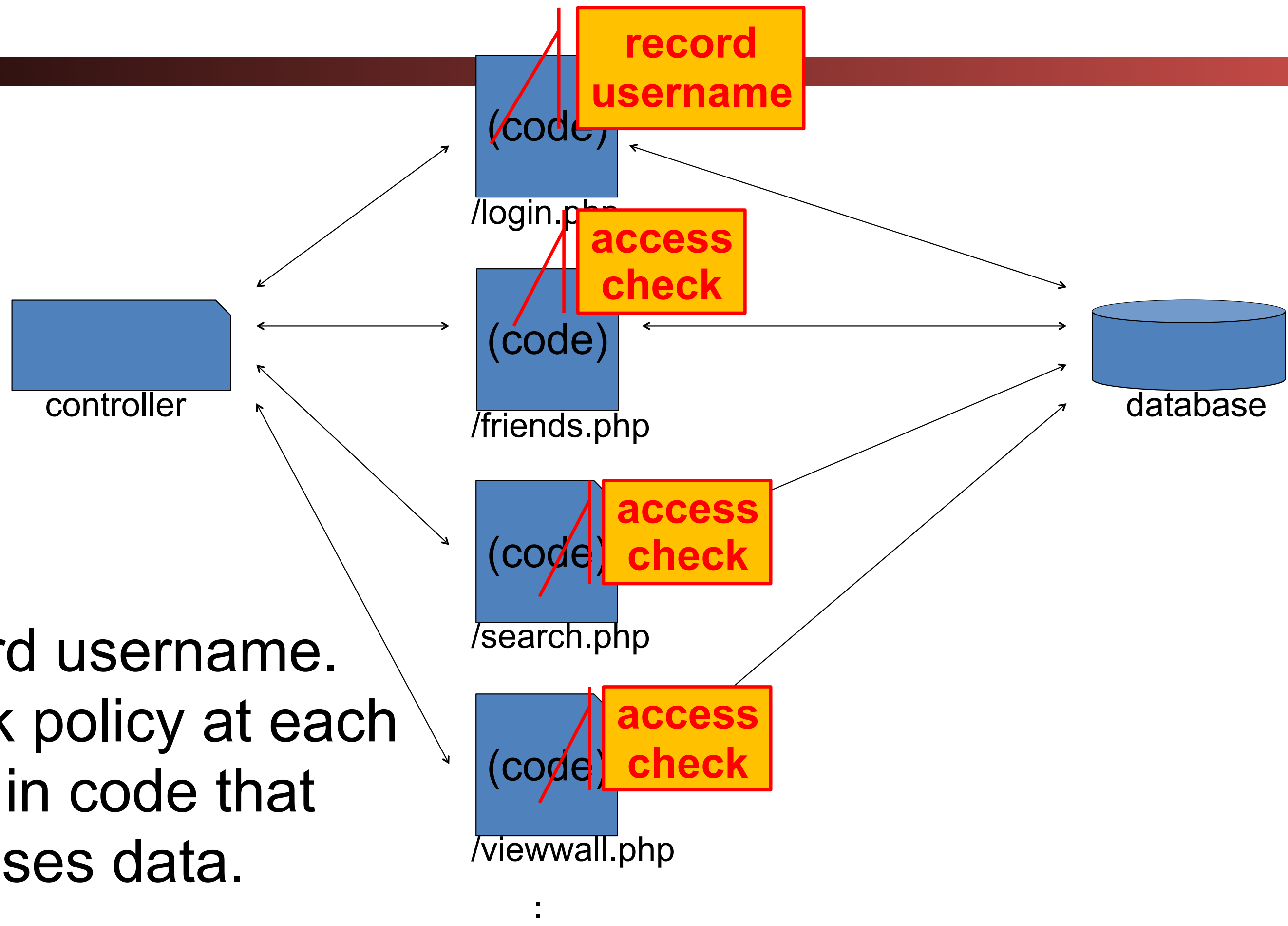
- Let’s talk about how this applies to web security…

Structure of a web application



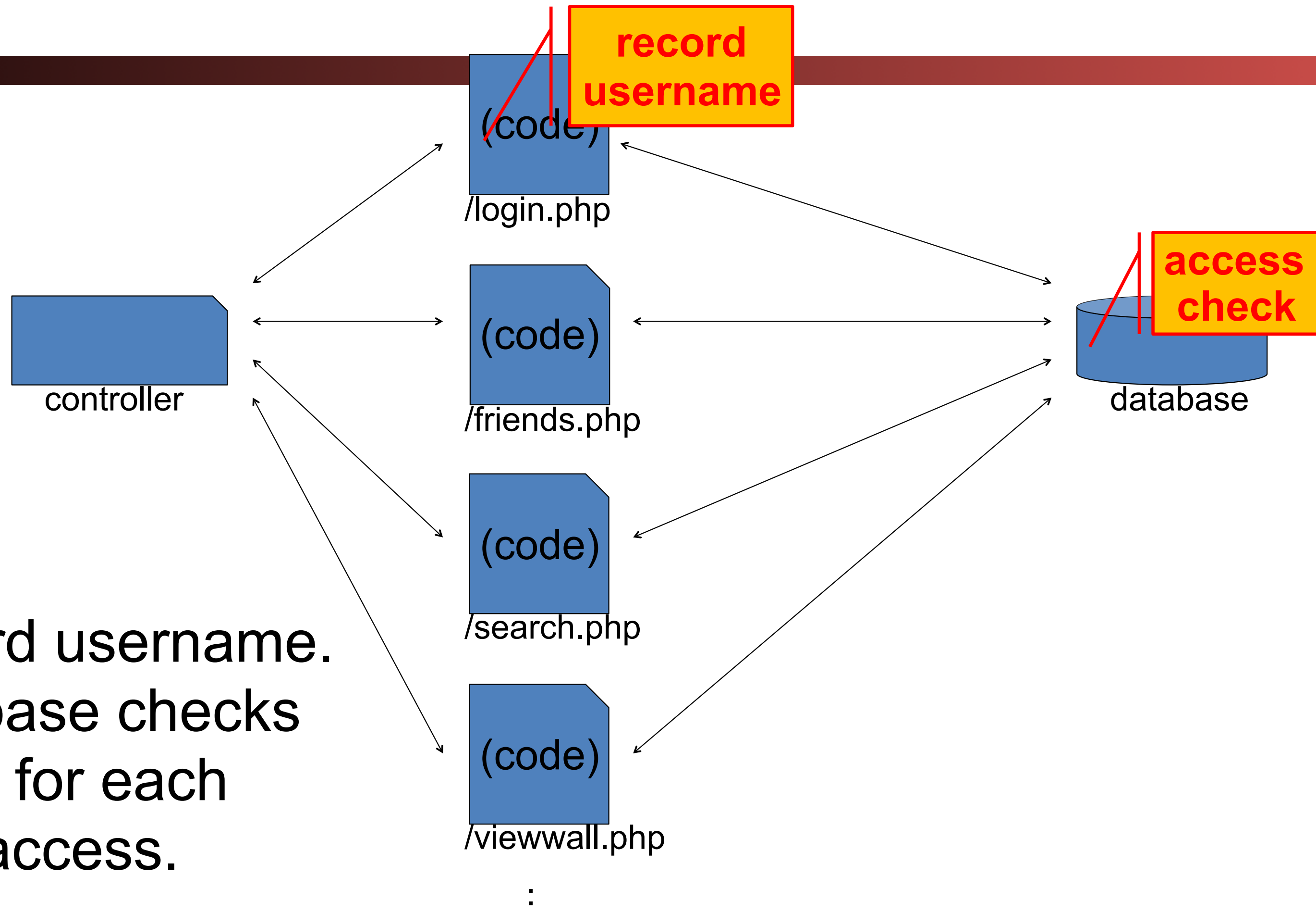
How should we implement access control policy?

Option 1: Integrated Access Control



Record username. Check policy at each place in code that accesses data.

Option 2: Centralized Enforcement



Record username. Database checks policy for each data access.

Analysis

- Centralized enforcement might be less prone to error
  - All accesses are vectored through a central chokepoint, which checks access
  - If you have to add checks to each piece of code that accesses data, it's easy to forget a check (and app will work fine in normal usage, until someone tries to access something they shouldn't)
- Integrated checks might be more flexible
  - But all it takes is missing ONE check to screw up!
- When in doubt, ***chose the more reliable option***

Access Control Groups

- Its often a pain to keep track of everyone individually
  - So instead lets create groups of people
- EG, "cs161-instructors", "cs161-students"
- This acts as a convenient shorthand
  - Now ***if*** we define access for a group and ***if*** we correctly identify who is in the group
- But groups also created of necessity for Unix access control

Unix/POSIX File Access Control: User/Group/All

- Unix and derivatives is ***old***
  - Development concepts date back to the late 1970s
  - ***Legacy*** often creates security problems and other issues
- In the old days, bits were expensive
  - Hard drives were measured in megabytes rather than terabytes
- Idea: each file entry has a small set of permission bits:
  - User/Group/All: Read/Write/Execute
    - Execute for programs means its runnable
    - Execute for folders means you can access files within it
      - But you need read to ***see*** files!
  - SUID/SETGID: When executed, run as the permissions of the file owner or the specified group

Windows File Access Control: ACLs

- ***Multi-user*** Windows is considerably newer with Windows-NT, 1993
  - By now, hard drives were starting to be measured in gigabytes
  - Microsoft's legacy problems are in a different area
- Microsoft uses Access Control Lists
  - Which can be arbitrarily long
- Each Access Control Entry (ACE) describes a user or group and the permissions allowed or denied
  - Also includes the notion of an "audit" permission noting that items need to be logged
- Uses the same mechanism for registry entries as well
- Apple's and Linux's file system also supports ACLs
  - Although naturally its a pain to use because the legacy stuff is still the common default for thinking about things

The "Superuser"

- In normal use, the user ***must not*** make changes that affect the system or other users
  - But sometimes you have to, well, fix things
- Enter the “Superuser”
  - An account with extra privileges
- Unix: “root”
- Windows: “Administrator”

Users and SUID programs

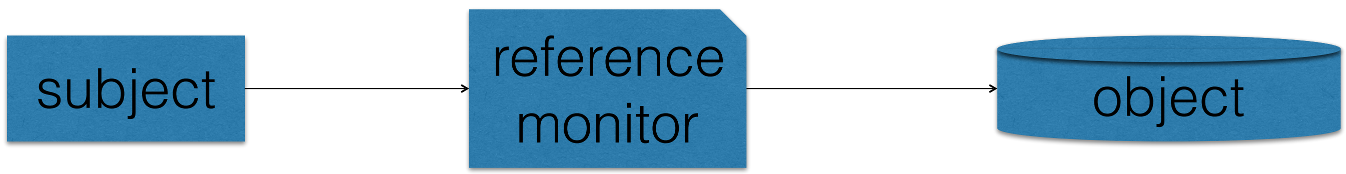
- A SUID program runs as the file's owner, not the invoking user
  - A very important property as it means it runs with the privileges of the file owner
- Many important things can only be done as the superuser "suid root"
  - Accept connections on low network ports
  - Become any ***other*** user
    - An important one being "nobody": the user with no additional permissions
- A vulnerability in a suid root program can generally compromise the entire machine

Complete mediation

- The principle: ***complete*** mediation
- Ensure that all access to data is mediated by something that checks access control policy.
  - In other words: the access checks can't be bypassed
- If you don't have complete mediation, your access control ***will*** fail!

Reference monitor

- A reference monitor is responsible for mediating all access to data



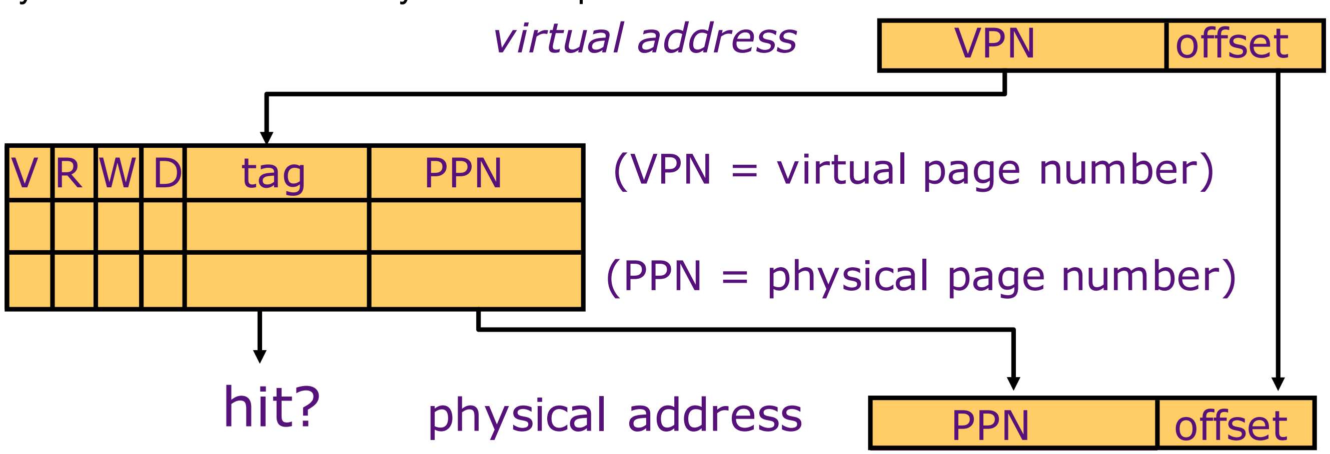
- Subject cannot access data directly; operations must go through the reference monitor, which checks whether they're OK

Criteria for a reference monitor

- Ideally, a reference monitor should be:
- ***Unbypassable***: all accesses go through the reference monitor
  - Otherwise an attacker will go around
- ***Tamper-resistant***: attacker cannot subvert or take control of the reference monitor (e.g., no code injection)
  - Otherwise an attacker will corrupt the reference monitor
- ***Verifiable***: reference monitor should be simple enough that it's unlikely to have bugs
  - Only small things can be validated reliably

One Such Reference Monitor: The processor's TLB

- Remember 61c: the Translation Lookaside Buffer
- When a program wishes to access memory:
  - If an entry exists and the operation is valid, adjust the address and allow
  - If no entry exists or the access type is invalid, trigger an interrupt
- When a program wishes to modify a TLB entry:
  - If CPU not in "kernel" mode, no updates are allowed
    - CPU can only enter "kernel" mode by an interrupt



Security Analysis and the TLB?

- Bypassable?
  - No. All program memory references must go through the TLB
- Tamper-Resistant?*
  - Yes. A program can not change any entries in the TLB: only kernel code can
- Verifiable?*
  - Yes. The TLB is relatively small hardware and is intensely verified
    - Hardware bugs are very costly so hardware designers are very comprehensive in testing systems

The Trusted Computing Base

- More broadly, the trusted computing base (TCB) is the subset of the system that has to be correct, for some security goal to be achieved
  - Example: the TCB for enforcing file access permissions includes the OS kernel and filesystem drivers
- Ideally, TCBs should be unbypassable, tamper-resistant, and verifiable
  - Which implies that TCBs are best when they are small:
    the more code -> the more you have to trust -> the more bugs

Ensuring Complete Mediation

- To secure access to some capability/resource, construct a reference monitor
- Single point through which all access must occur
  - E.g.: a network firewall
- Desired properties:
  - Un-bypassable ("complete mediation")
  - Tamper-proof (is itself secure)
  - Verifiable (correct)
  - (Note, just restatements of what we want for TCBs)
- One subtle form of reference monitor flaw concerns race conditions …

So about that *

- The Trusted Base for correct memory access is ***not just the TLB***
  - Thus the trusted base is considerably larger (and therefore considerably weaker)
- The TLB relies on two other things:
  - The CPU ***must not*** go into kernel mode except when an interrupt occurs
    - This is probably a reasonable assumption…
  - The OS kernel ***must not*** allow any non-kernel code to execute in the kernel or allow it to change the state of the kernel's memory mappings
    - This is a much harder assumption

TCBs in Practice: Apple iPhones

- The iPhone actually has multiple TCBs for different purposes:
  - The fingerprint sensor
  - The “Secure Enclave” cryptographic engine
  - The more general OS
- Each TCB trades-off the complexity of what it protects vs the security of what it protects
  - Its far easier to build a TCB that just does a little thing

The Fingerprint Sensor

- Desired property: ***only*** the untampered fingerprint reader communicates to the secure enclave
  - Don't allow someone to replace it with one which can replay a fingerprint
- The home button's fingerprint sensor has very limited functionality
  - When the phone is created, it establishes a secured channel to the "Secure Enclave"
  - A new fingerprint reader can be replaced, but only by Apple as it requires telling the device to accept a new reader using a key only Apple possess

The Secure Enclave

- A separate processor running in the chip
  - Has exclusive access to a random device key created during manufacturing
- Handles all the cryptography and authentication
  - A very limited window for communication with the main processor
  - The fingerprint reader is forwarded from the main processor
    - But that communication is encrypted with a key the main processor doesn't know
- Goal is very strong but very limited:
  - Protect the encryption keys used to store data so that w/o the password the data is inaccessible
  - Authenticate for payment systems (Apple Pay)

The General iOS Kernel

- The “kernel” on the phone is the primary operating system
  - It does ***not*** have access to the cryptography engine, but can only make requests to enable decryption of memory
- But it does have complete control over the rest of the phone
- If the phone is locked:
  - Kernel doesn’t have access to encrypted data
- If the phone is ***unlocked***:
  - Kernel can read/write all the encrypted data ***even though it doesn’t have the key***
    - But can’t process payment requests

Optional Reading (For Now): Apple iOS security guide

- Linked to on the course webpage…
- For ***now***, just look through the part on TouchID and Secure Enclave
- But by the end of the course, the entire document ***will*** become required reading
  - Its a great test of your understanding of security concepts:
    Why does Apple do what they do?
    What would you do differently?
    What tradeoffs are involved?

Robustness

- Security bugs are a fact of life
- How can we use access control to improve the security of software, so security bugs are less likely to be catastrophic?

Privilege separation

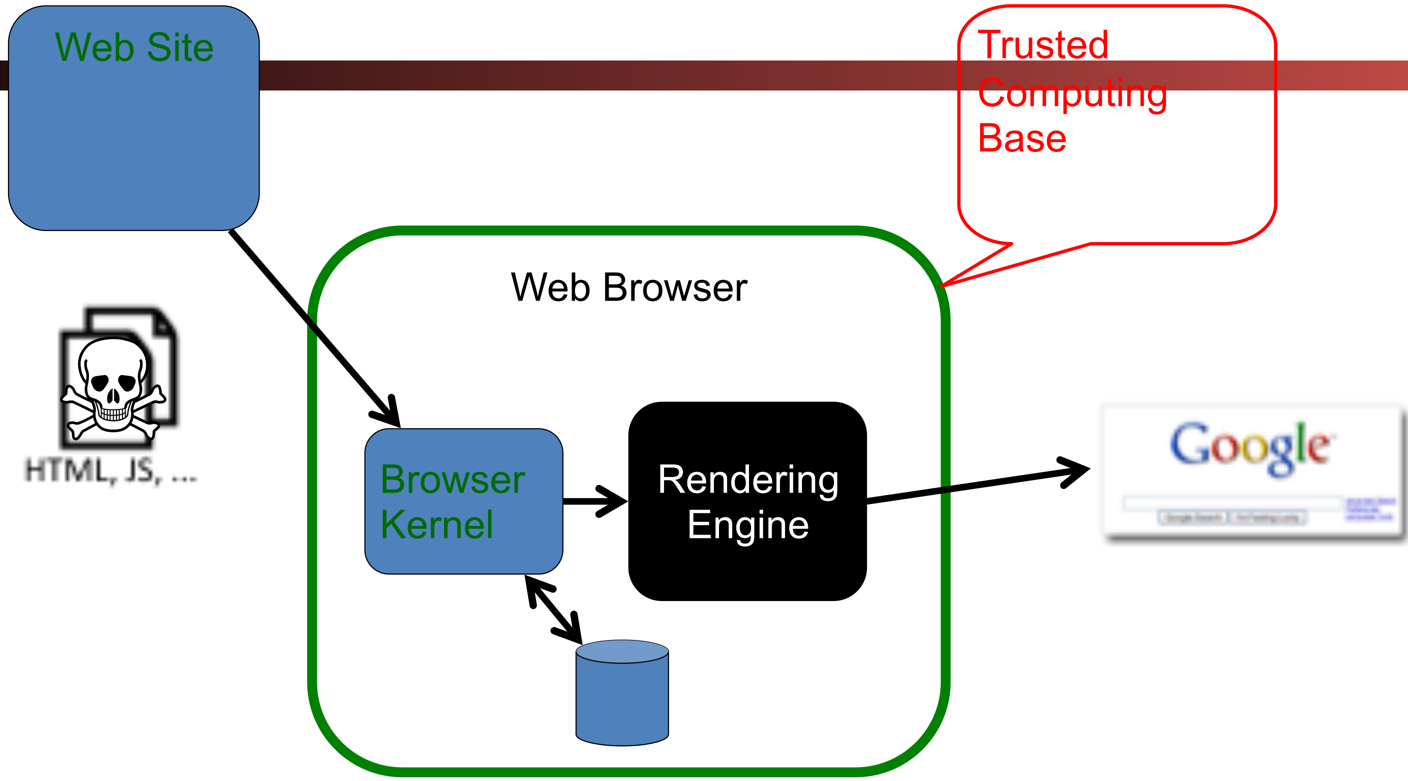
- How can we improve the security of software, so security bugs are less likely to be catastrophic?

- Answer: privilege separation.
  Architect the software so it has a separate, small TCB.
  – Then any bugs outside the TCB will not be catastrophic

Touchstones for Least Privilege

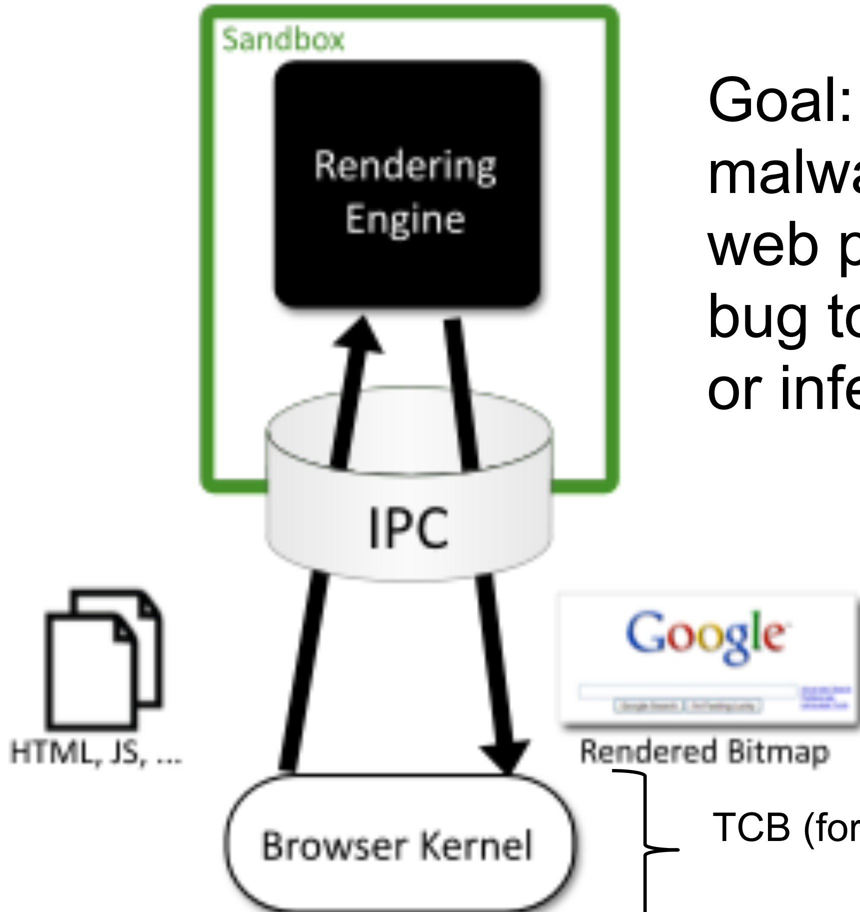
- When assessing the security of a system's design, identify the Trusted Computing Base (TCB).
  - What components does security rely upon?
- Security requires that the TCB:
  - Is correct
  - Is complete (can't be bypassed)
  - Is itself secure (can't be tampered with)
- Best way to be assured of correctness and its security?
  - KISS = Keep It Simple, Stupid!
  - Generally, Simple = Small
- One powerful design approach: privilege separation
  - Isolate privileged operations to as small a component as possible
  - (See lecture notes for more discussion)

Web browser



"Drive-by malware": malicious web page exploits a browser bug to read/write local files or infect them with a virus

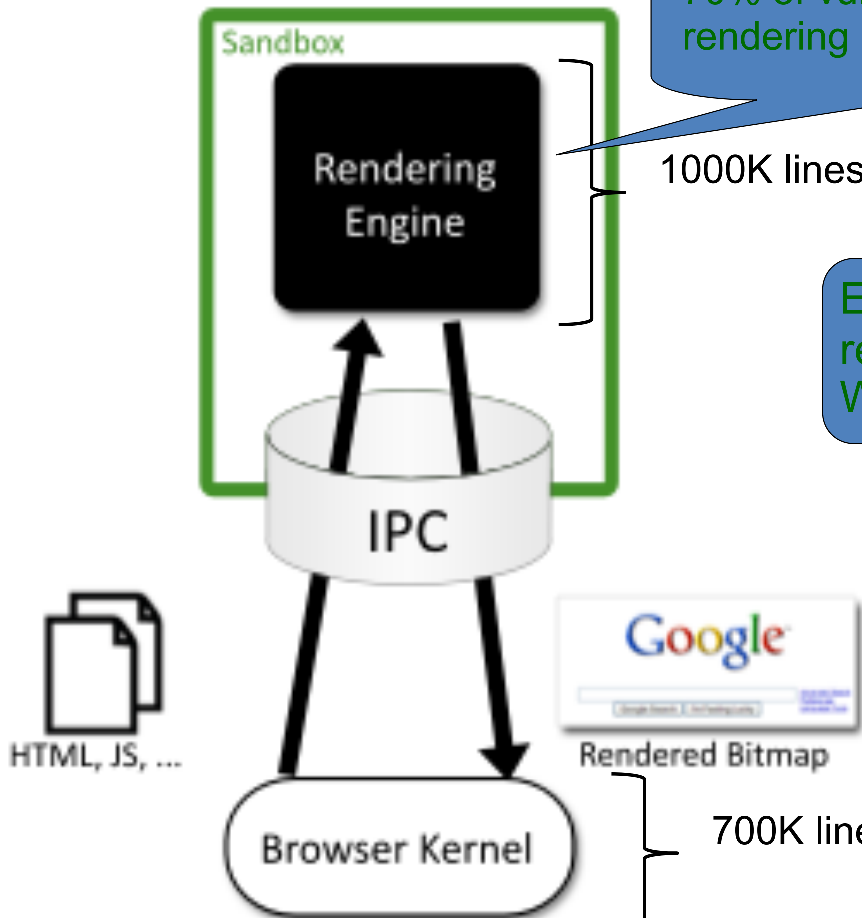
The Chrome browser



Goal: prevent “drive-by malware”, where a malicious web page exploits a browser bug to read/write local files or infect them with a virus

TCB (for this property)

The Chrome browser



70% of vulnerabilities are in the rendering engine.

1000K lines of code

Example: PNG, WMF, GDI+ rendering vulnerabilities in Windows OS

700K lines of code

Constructing Sandboxes

- Need to provide a constrained communication mechanism
  - A clean API to separate the sandboxed elements
- Need a mechanism to ***give up*** privileges
  - So that the sandboxed component ***can not*** do things outside the sandbox
- In the end it is really more of a ***litterbox***
  - But an attacker needs to both compromise the program in the sandbox ***and*** escape from the sandbox to impact the program

Time of Check To Time of Use (TOCTTOU)

- A ***very*** common class of bugs in a reference monitor
  - Check to see if an action is allowed
  - Perform that action
- But somewhere in between the check and use, conditions are changed
  - So it would no longer be allowed
- Most attacks are ***race conditions***:
  - Attacker needs to win the “race” to change conditions after the check but before the action happens

Exploiting TOCTTOU: Race Conditions

- Lets take a simple SUID root program:
  - Check if user should be allowed to write to a particular file
  - Open the file for writing
- But what if the file is a link and the attacker changes the file?
  - Can use this to overwrite anything… such as the **`/etc/sudoers`** file

```
if (!access_ok(file)
  abort();
open(file);
write(file);
```

Preventing TOCTTOU: Atomicity

- Robustly preventing TOCTTOU ***requires*** some form of atomicity
  - Either a way of locking things so that changes can't happen
  - OR an exception mechanism that does the check atomically
    - EG, a SUID program temporarily changes who its running to using **`seteuid`** and then calling **`open`** directly
- Otherwise, you always have these problems
- A consequence: the Unix **`access()`** function is completely broken
  - Its intent: Can the process calling the current SUID program also access the file?
  - Its result: Using access it is ***impossible*** to provably prevent TOCTTOU errors!