# CE 874 - Secure Software Systems

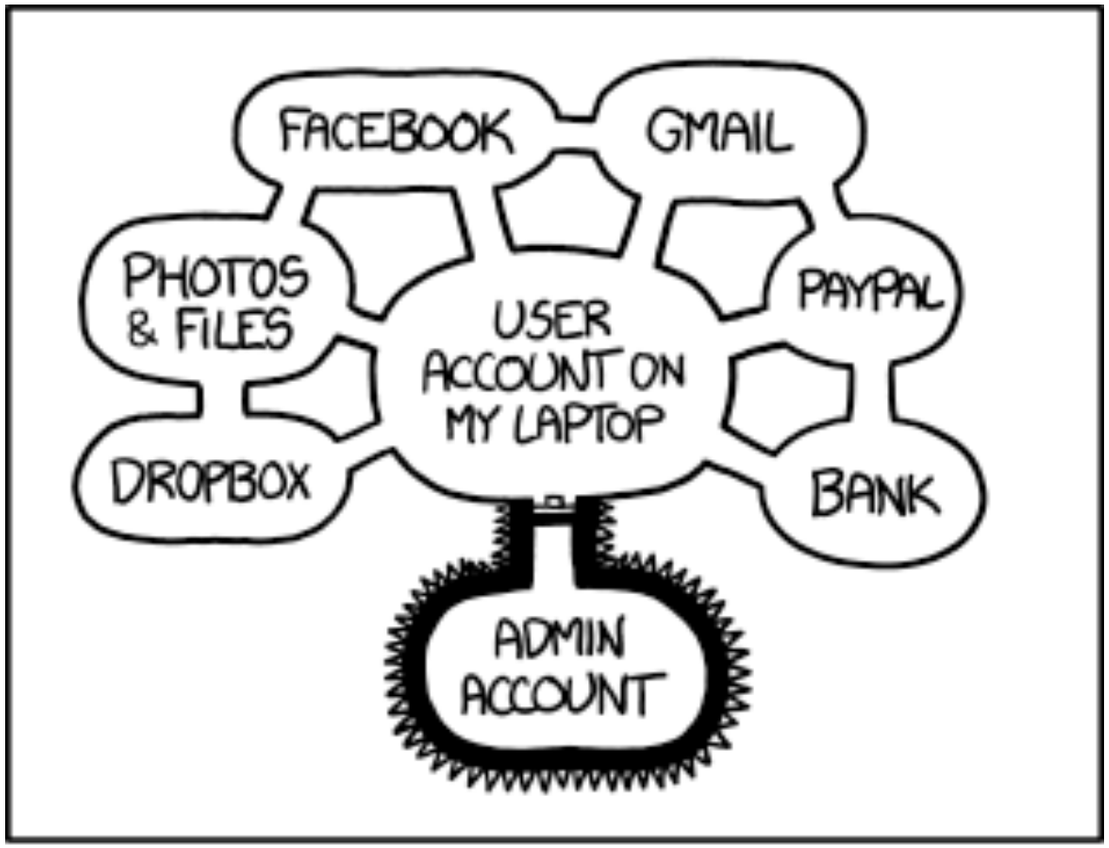
Secure Architecture I

Mehdi Kharrazi
Department of Computer Engineering
Sharif University of Technology

*Acknowledgments*: Some of the slides are fully or partially obtained from other sources. A reference is noted on the bottom of each slide, when the content is fully obtained from another source. Otherwise a full list of references is provided on the last slide.

# Secure Architecture

- How to come up with a secure architecture?
- What design principals is should be followed?
- What are the available mechanisms?
- How do you trust the code getting executed?

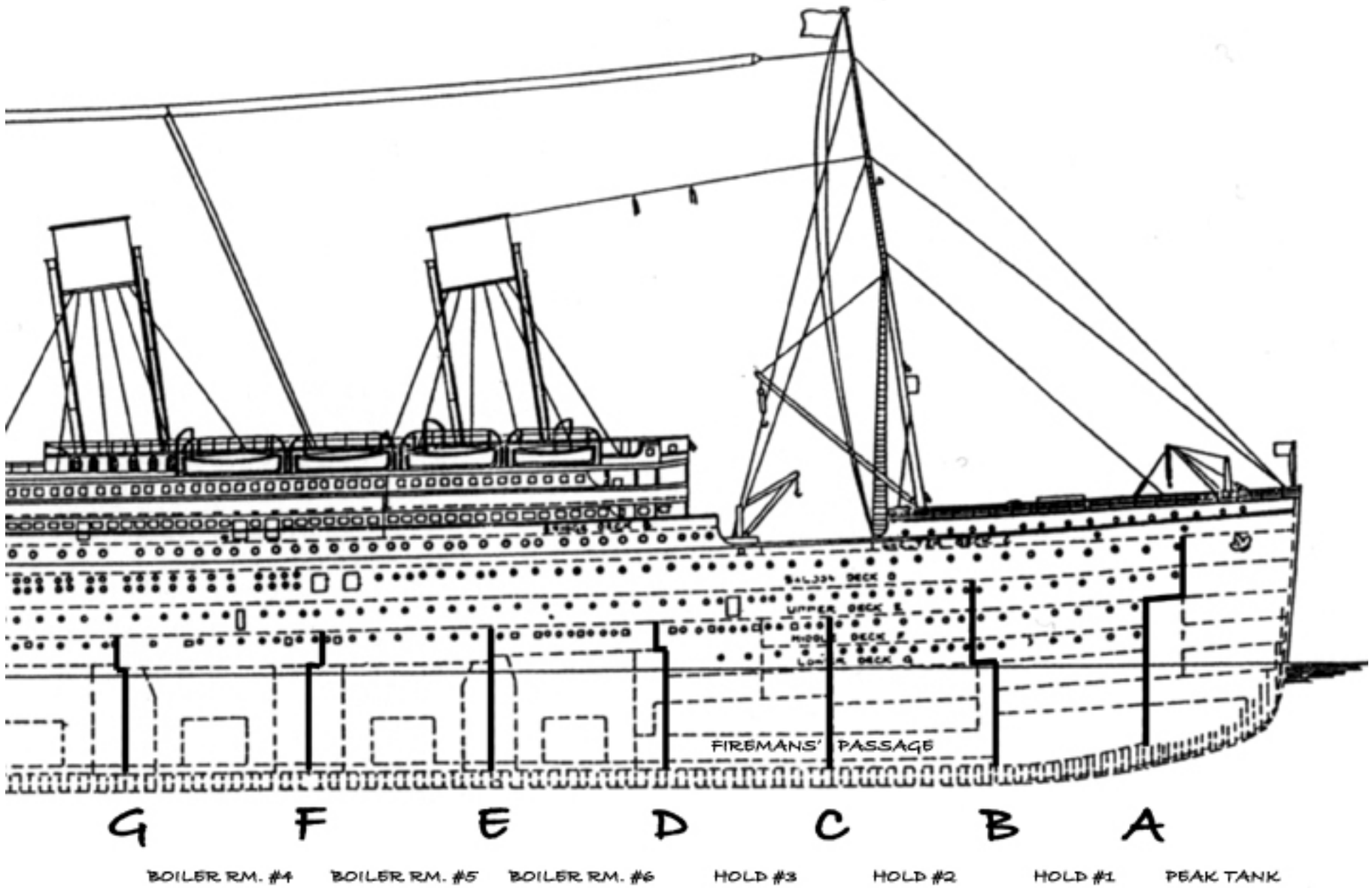



IF SOMEONE STEALS MY LAPTOP WHILE I'M LOGGED IN, THEY CAN READ MY EMAIL, TAKE MY MONEY, AND IMPERSONATE ME TO MY FRIENDS,

BUT AT LEAST THEY CAN'T INSTALL DRIVERS WITHOUT MY PERMISSION. xkcd.com

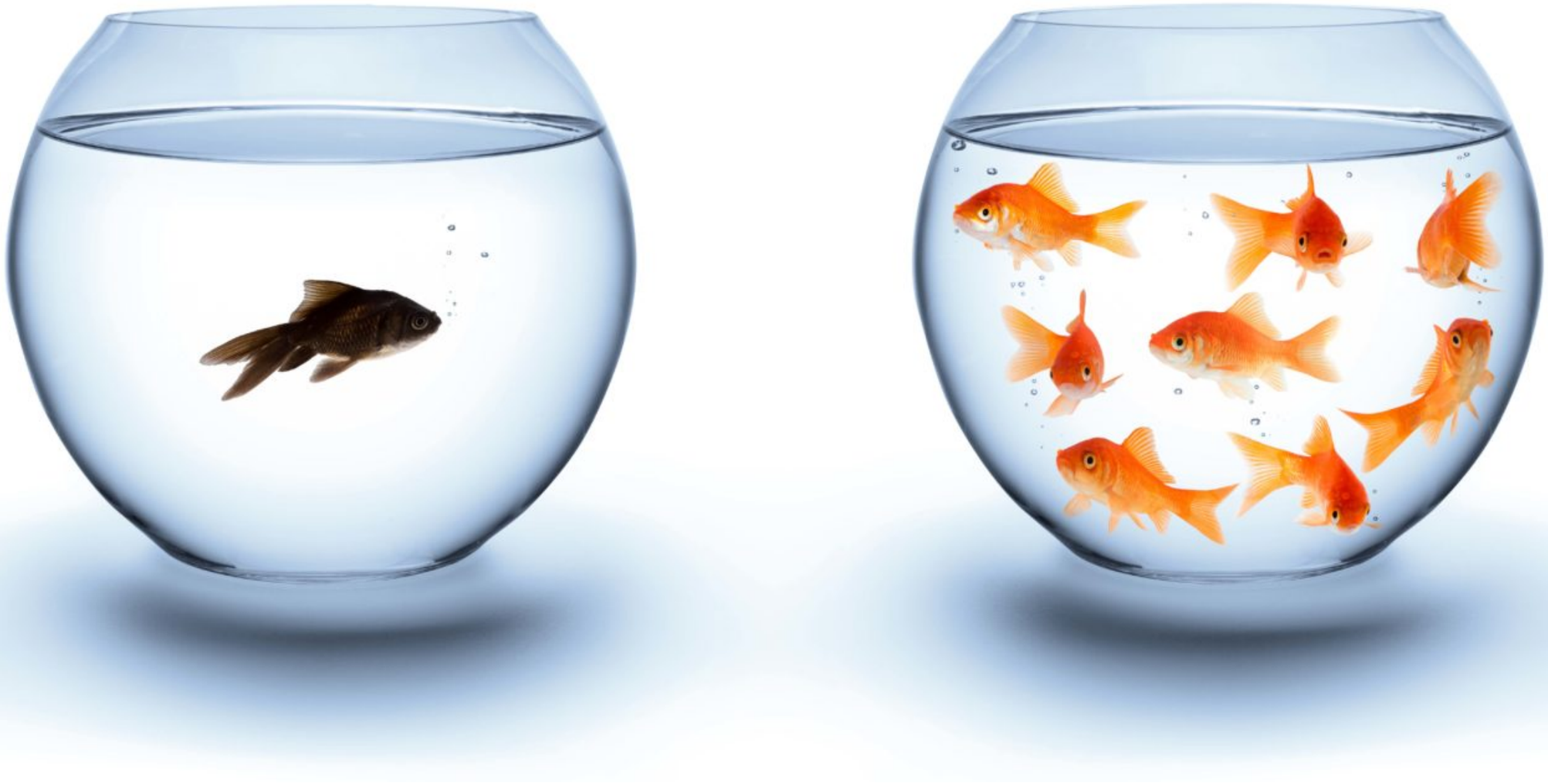
# Building a Secure system


okieboat.com




imsa.edu

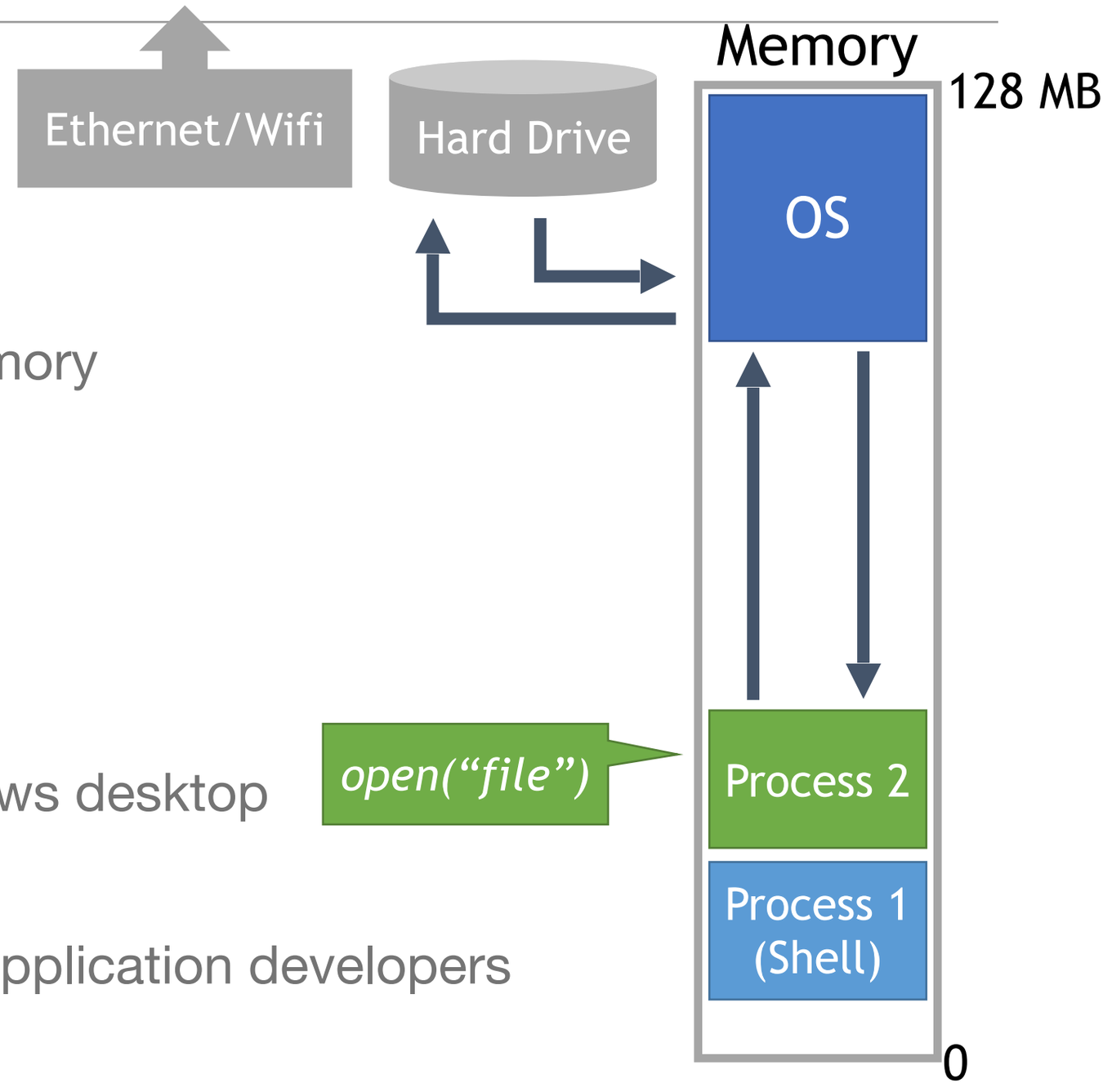
# Isolation



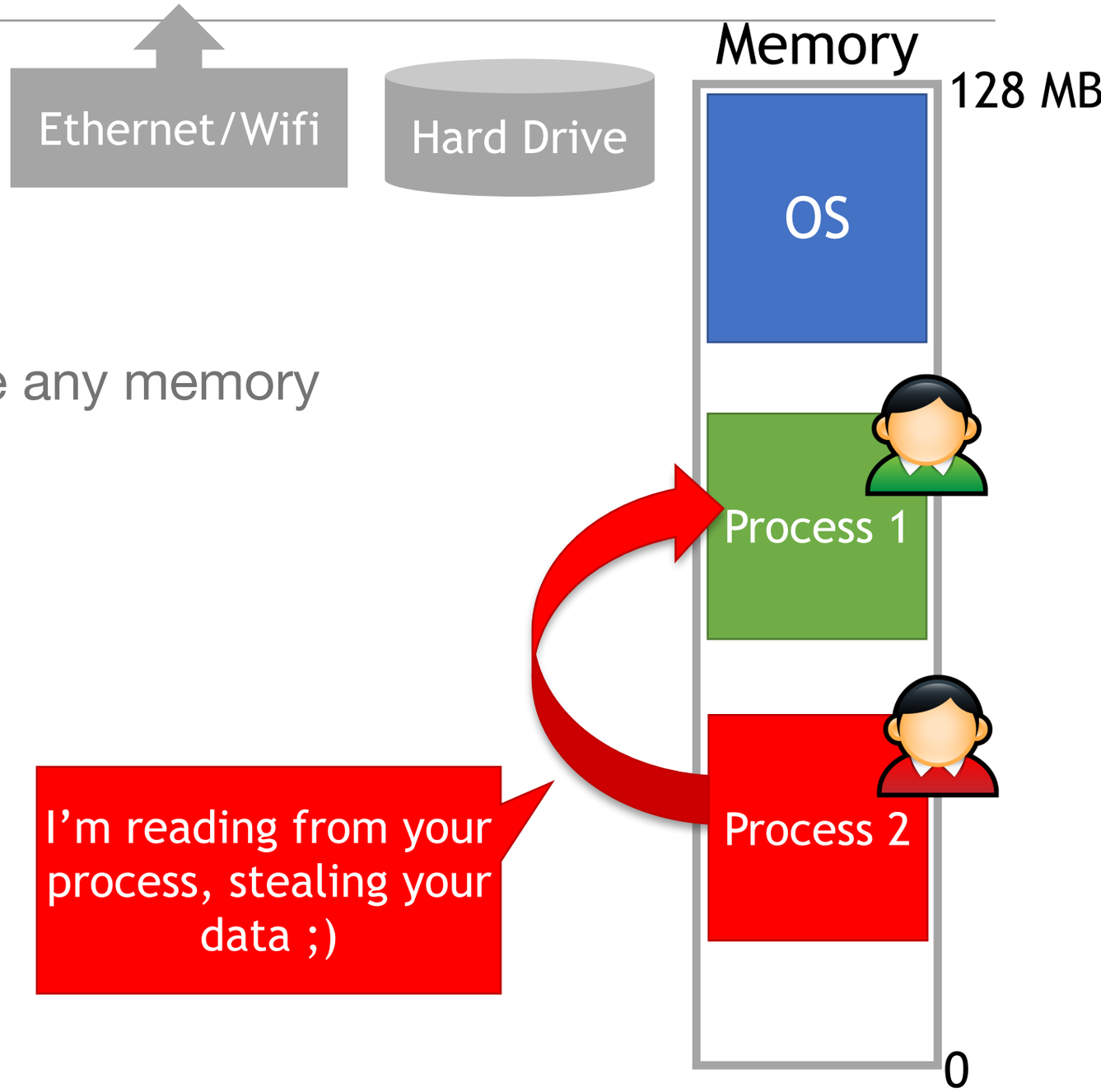

IBM
IBM
IBM

# System Model

- On bootup, the Operating System (OS) loads itself into memory
  - DOS or Windows 3.1
  - Typically places itself in high memory
- What is the role of the OS?
  - Allow the user to run processes
  - Often comes with a shell
    - Text shell like bash
    - Graphical shell like the Windows desktop
  - Provides APIs to access devices
    - Offered as a convenience to application developers

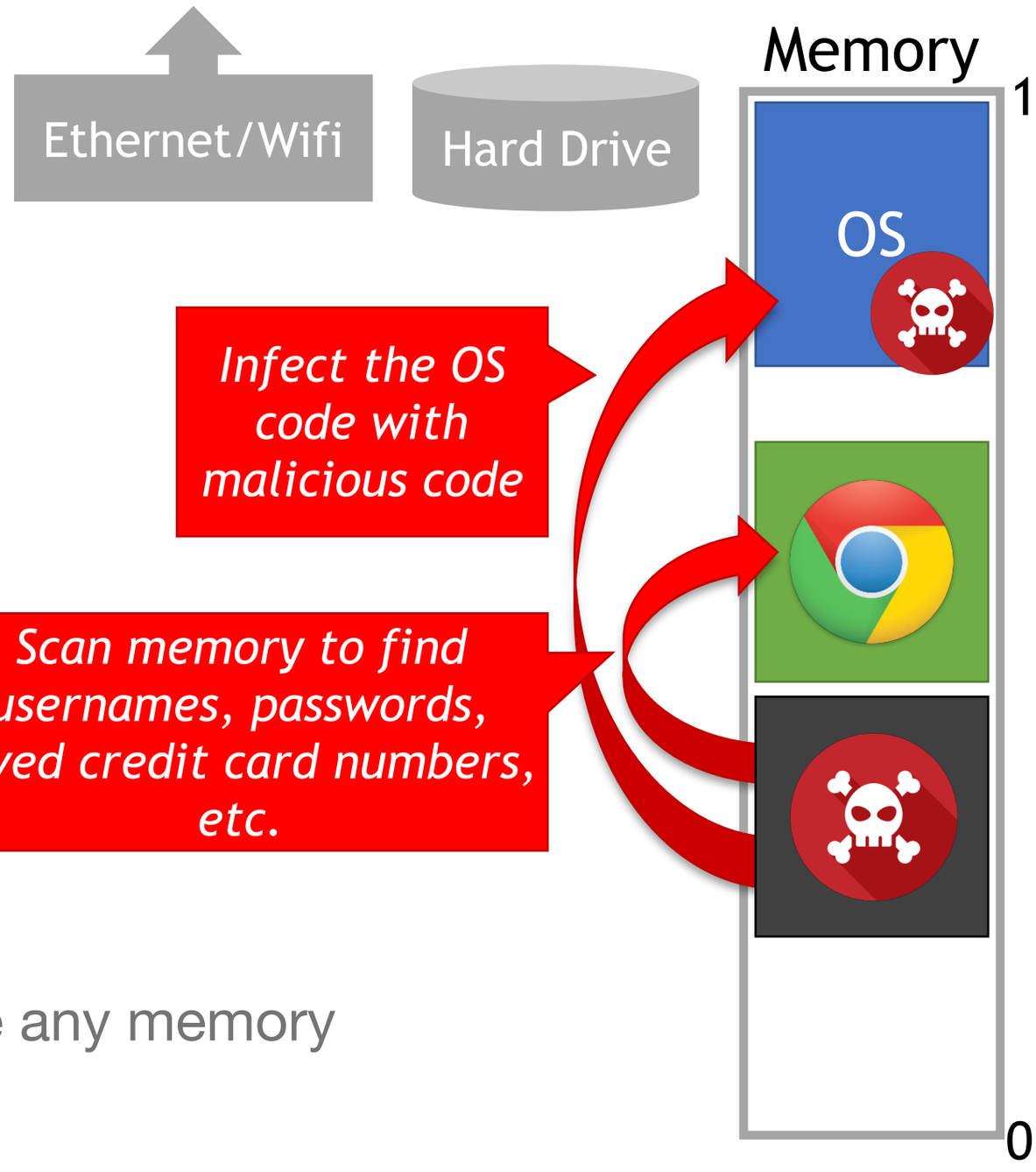


# Memory Unsafety




- Problem: any process can read/write any memory

# Memory Unsafety



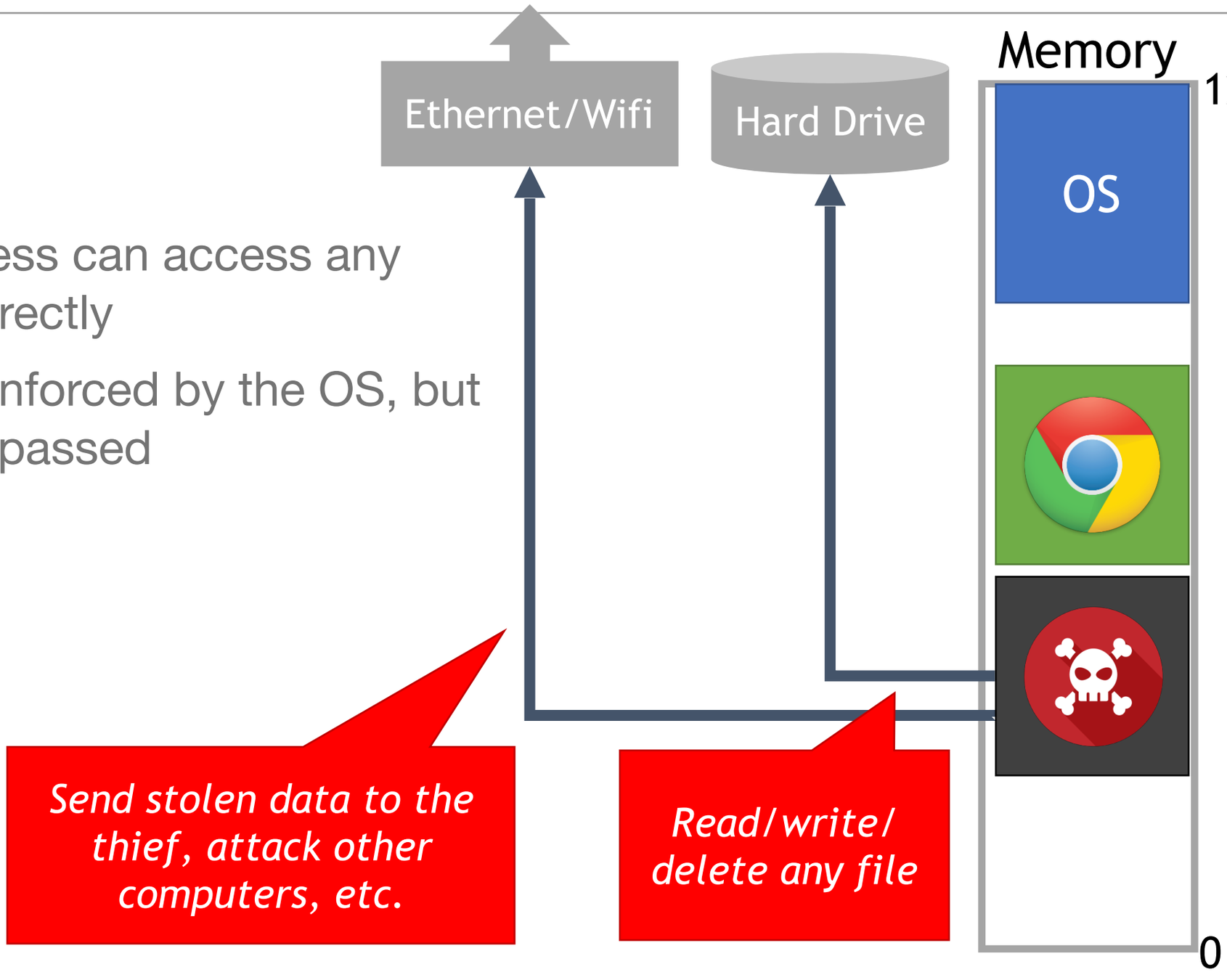

- Problem: any process can read/write any memory

# Device Unsafety

- Problem: any process can access any hardware device directly
- Access control is enforced by the OS, but OS APIs can be bypassed

# Device Unsafety

- Problem: any process can access any hardware device directly
- Access control is enforced by the OS, but OS APIs can be bypassed



# Older system issues

- Old systems did not protect memory or devices
  - Any process could access any memory
  - Any process could access any device
- Problems
  - No way to enforce access controls on users or devices
    - Processes can steal from or destroy each other
    - Processes can modify or destroy the OS

- On old computers, systems security was literally impossible

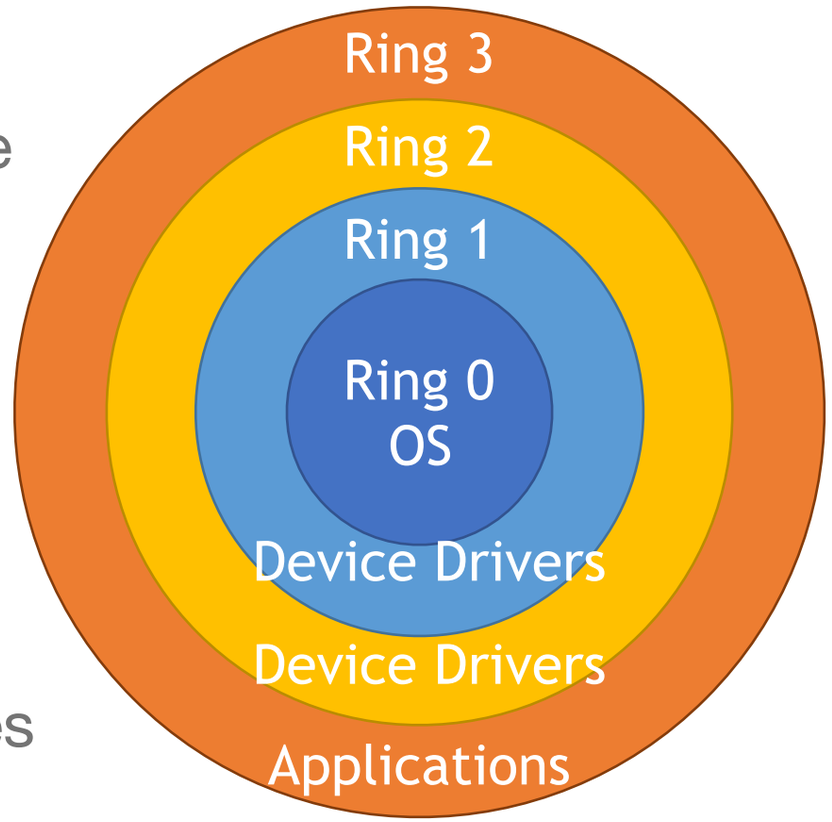
# Hardware support for isolation

# Towards Modern Architecture

- To achieve systems security, we need process isolation
  - Processes cannot read/write memory arbitrarily
  - Processes cannot access devices directly
- How do we achieve this?
- Hardware support for isolation
  - Protected mode execution (a.k.a. process rings)
  - Virtual memory

# Protected Mode

- Most modern CPUs support protected mode
- x86 CPUs support three rings with different privileges
  - Ring 0: Operating System
    - Code in this ring may directly access any device
  - Ring 1, 2: device drivers
    - Code in these rings may directly access some devices
    - May not change the protection level of the CPU
  - Ring 3: userland
    - Code in this ring may not directly access devices
    - All device access must be via OS APIs
    - May not change the protection level of the CPU
- Most OSes only use rings 0 and 3



# System Boot Sequence

- On startup, the CPU starts in 16-bit real mode
  - Protected mode is disabled
  - Any process can access any device
- BIOS executes, finds and loads the OS
- OS switches CPU to 32-bit protected mode
  - OS code is now running in Ring 0
  - OS decides what Ring to place other processes in
- Shell gets executed, user may run programs
  - User processes are placed in Ring 3

# Restriction on Privileged Instructions

- What CPU instructions are restricted in protected mode?
  - Any instruction that modifies the CR0 register
    - Controls whether protected mode is enabled
  - Any instruction that modifies the CR3 register
    - Controls the virtual memory configuration
  - hlt – Halts the CPU
  - sti/cli – enable and disable interrupts
  - in/out – directly access hardware devices
- If a Ring 3 process tries any of these things, it immediately crashes

# Changing Modes

- Applications often need to access the OS APIs
  - Writing files
  - Displaying things on the screen
  - Receiving data from the network
  - etc…
- But the OS is Ring 0, and processes are Ring 3
- How do processes get access to the OS?
  - Invoke OS APIs with special assembly instructions
    - Interrupt: int 0x80
    - System call: sysenter or syscall
  - int/sysenter/syscall cause a mode transfer from Ring 3 to Ring 0

# Mode Transfer

- Application executes trap (int) instruction
  - EIP, CS, and EFLAGS get pushed onto the stack
  - Mode switches from ring 3 to ring 0
- Save the state of the current process
  - Push EAX, EBX, …, etc. onto the stack
- Locate and execute the correct syscall handler
- Restore the state of process
  - Pop EAX, EBX, … etc.
- Place the return value in EAX
- Use iret to return to the process
  - Switches back to the original mode (typically 3)

# Virtual Memory Implementation

- Each process has its own virtual memory space
  - Each process has a page table that maps is virtual space into physical space
  - CPU translates virtual address to physical addresses on-the-fly
- OS creates the page table for each process
  - Installing page tables in the CPU is a protected, Ring 0 instruction
  - Processes cannot modify their page tables
- What happens if a process tries to read/write memory outside its page table?
  - Segmentation Fault or Page Fault
  - Process crashes
  - In other words, no way to escape virtual memory

# Security Policy in General

# Security Policy

- Defines a security perimeter

Because you can't secure everything

# Security Policy

- Defines a security perimeter
- Standards codify the what should be done
- Guidelines explain how it will be done

# How do you create a policy?

- Option #1 Risk Assessment:
  - Identify assets and their value
  - Identify the threats
  - Calculate the risks
  - Conduct a Cost-Benefit Analysis
- Option #2: Adopt “Best Practices.”

# Threat Modeling

- Threat modeling is the process of systematically identifying the threats faced by a system

- Identify things of value that you want to protect
- Enumerate the attack surfaces
- Hypothesize attackers and map them to
  - Things of value they want from (1)
  - Their ability to target vulnerable surfaces from (2)
- Survey mitigations
- Balance costs versus risks

# Identify Things of Value

- Saved passwords
- Monetizable credentials (webmail, social networks)
- Access to bank accounts, paypal, venmo, credit cards, or other financial services
- Pics, messages, address book, browsing/search history (for blackmail)
- Sensitive business documents
- Access to sensors (camera, mic, GPS) or network traffic (for surveillance)
- The device itself
  - Steal it and sell it
  - Use the CPU and network for other criminal activity

# Enumerate Attack Surfaces

- Intercept and compromise the handset in transit
- Backdoor the OS
- Steal the device and use it
- Direct connection via USB
- Close proximity radios (Bluetooth, NFC)
- Social engineering, e.g. trick the user into installing malicious app(s)
- Exploit vulnerabilities in the OS or apps (e.g. email clients, web browsers)
- Passive eavesdropping on the network
- Active network attacks (e.g. man-in-the-middle, SMS of death)

# Techniques For Drafting Policies

- Assign a specific “owner” to everything that is to be protected.
- Be positive
- Be realistic in your expectations
- Concentrate on education and prevention

# Remember, Risk Cannot Be Eliminated

- You can purchase a UPS…
  - But the power failure may outlast the batteries
  - But the UPS may fail
  - But the cleaning crew may unplug it
  - But the UPS may crash due to a software error.

# Spaf's first principle of security administration:

"If you have responsibility for security, but have no authority to set rules or punish violators, your own role in the organization is to take the blame when something big goes wrong."

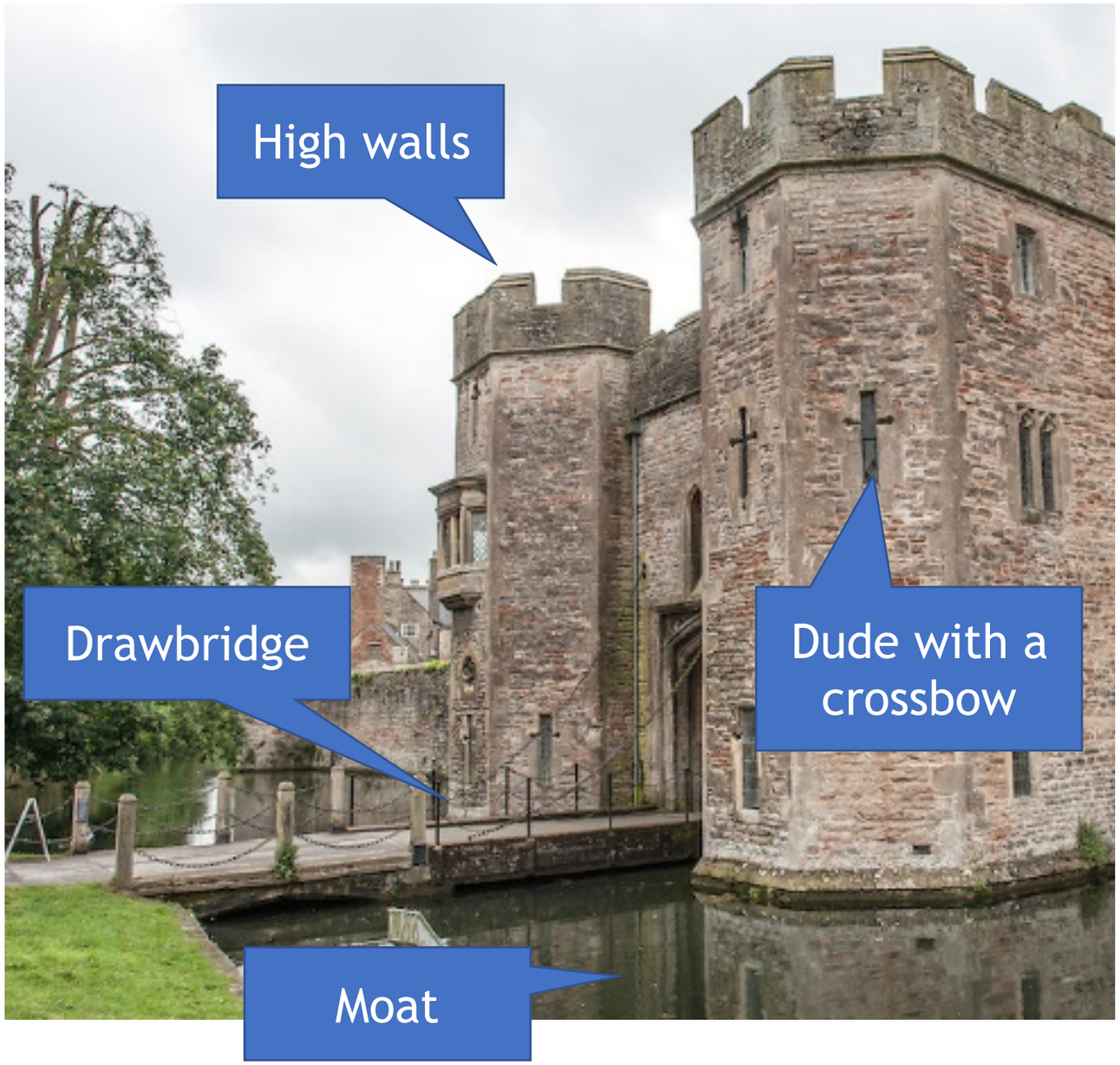
# Security Principles

# Security Principles

- Designing secure systems (and breaking them) remains an art
- Security principles help bridge the gap between art and science
  - Developed by Saltzer and Schroeder
  - “The Protection of Information in Computer Systems”, 1975

# Defense in Depth

- Don't depend on a single protection mechanism, since they are apt to fail
- Even very simple or formally verified defenses fail
- Layering defenses increases the difficulty for attackers
- Defenses should be complementary!



# Defence in Depth Example

- Problem: Bank.
  - How to secure the money?
- Solution: Defence in depth.
  - Guards inside bank.
  - Closed-circuit cameras monitor activity.
  - Tellers do not have access to vault.
  - Vault has multiple defences:
    - Time-release.
    - Walls and lock complexity.
    - Multiple compartments.

# Example

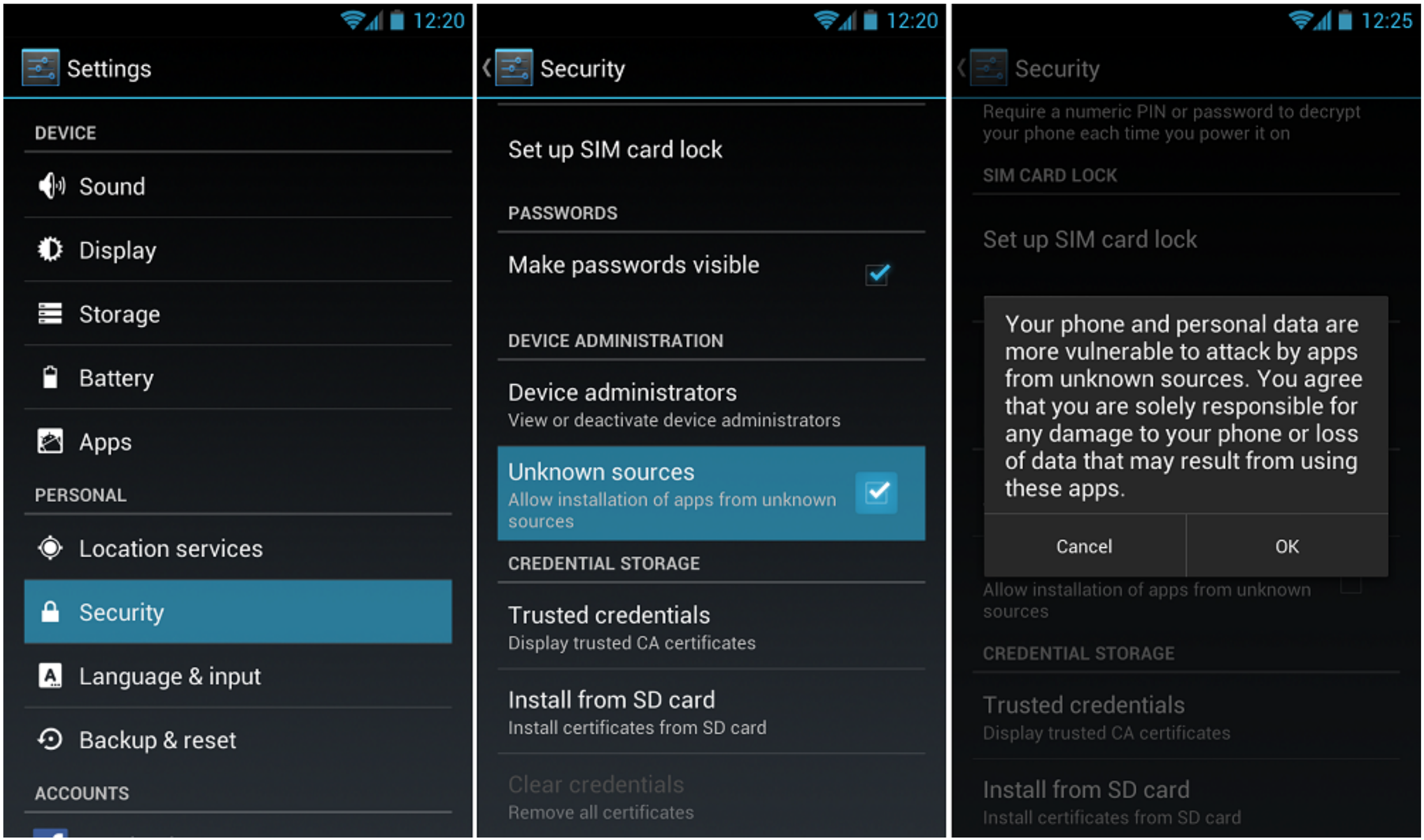
- Built-in security features of Windows 10
  - Secure boot: cryptographically verified bootup process
  - Bitlocker full-drive encryption
  - Kernel protections, e.g. Address Space Layout Randomization (ASLR)
  - Cryptographic signing for device drivers
  - User authentication
  - User Account Control: permission check for privileged operations
  - Anti-virus and anti-malware
  - Firewall
  - Automated patching
  - System logs

# Fail-safe Defaults

- The absence of explicit permission is equivalent to no permission

- Systems should be secure "out-of-the-box"
  - Most users stick with defaults
  - Users should "opt-in" to less-secure configurations


- Examples. By default…
  - New user accounts do not have admin or root privileges
  - New apps cannot access sensitive devices
  - Passwords must be >8 characters long
  - Etc.


12:20
Settings
DEVICE
Sound
Display
Storage
Battery
Apps
PERSONAL
Location services
Security
Language & input
Backup & reset
ACCOUNTS
12:20
Security
Set up SIM card lock
PASSWORDS
Make passwords visible
DEVICE ADMINISTRATION
Device administrators
View or deactivate device administrators
Unknown sources
Allow installation of apps from unknown sources
CREDENTIAL STORAGE
Trusted credentials
Display trusted CA certificates
Install from SD card
Install certificates from SD card
Clear credentials
Remove all certificates
12:25
Security
Require a numeric PIN or password to decrypt your phone each time you power it on
SIM CARD LOCK
Set up SIM card lock
Your phone and personal data are more vulnerable to attack by apps from unknown sources. You agree that you are solely responsible for any damage to your phone or loss of data that may result from using these apps.
Cancel
OK
Allow installation of apps from unknown sources
CREDENTIAL STORAGE
Trusted credentials
Display trusted CA certificates
Install from SD card
Install certificates from SD card

[Wilson'19]

# Separation of Privilege

- Privilege, or authority, should only be distributed to subjects that require it
- Some components of a system should be less privileged than others
  - Not every subject needs the ability to do everything
  - Not every subject is deserving of full trust

- Examples:
  - Two signatures required for a check
  - Two authorized personnel required to fire a nuclear missile

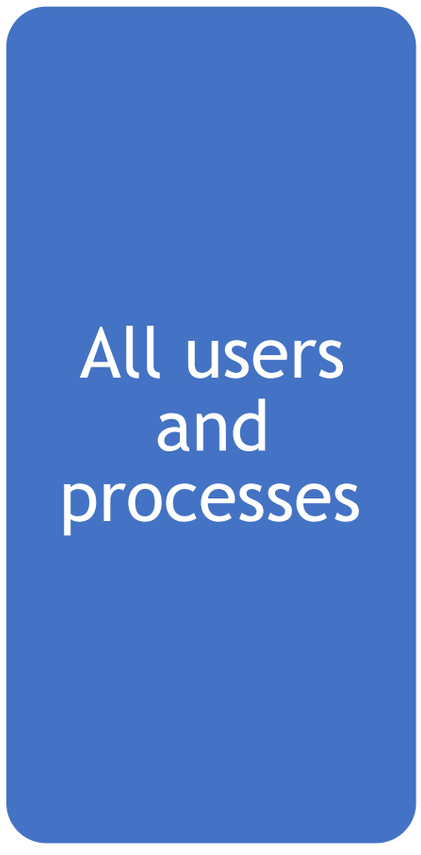
# Least Privilege

- Subjects should possess only that authority that is required to operate successfully

- Closely related to separation of privilege
- Not only should privilege be separated, but subjects should have the least amount necessary to perform a task

# Privilege Over Time

DOS, Windows 3.1




Win 95 and 98


Win NT, XP, 7, 8, 10
Linux, BSD, OSX

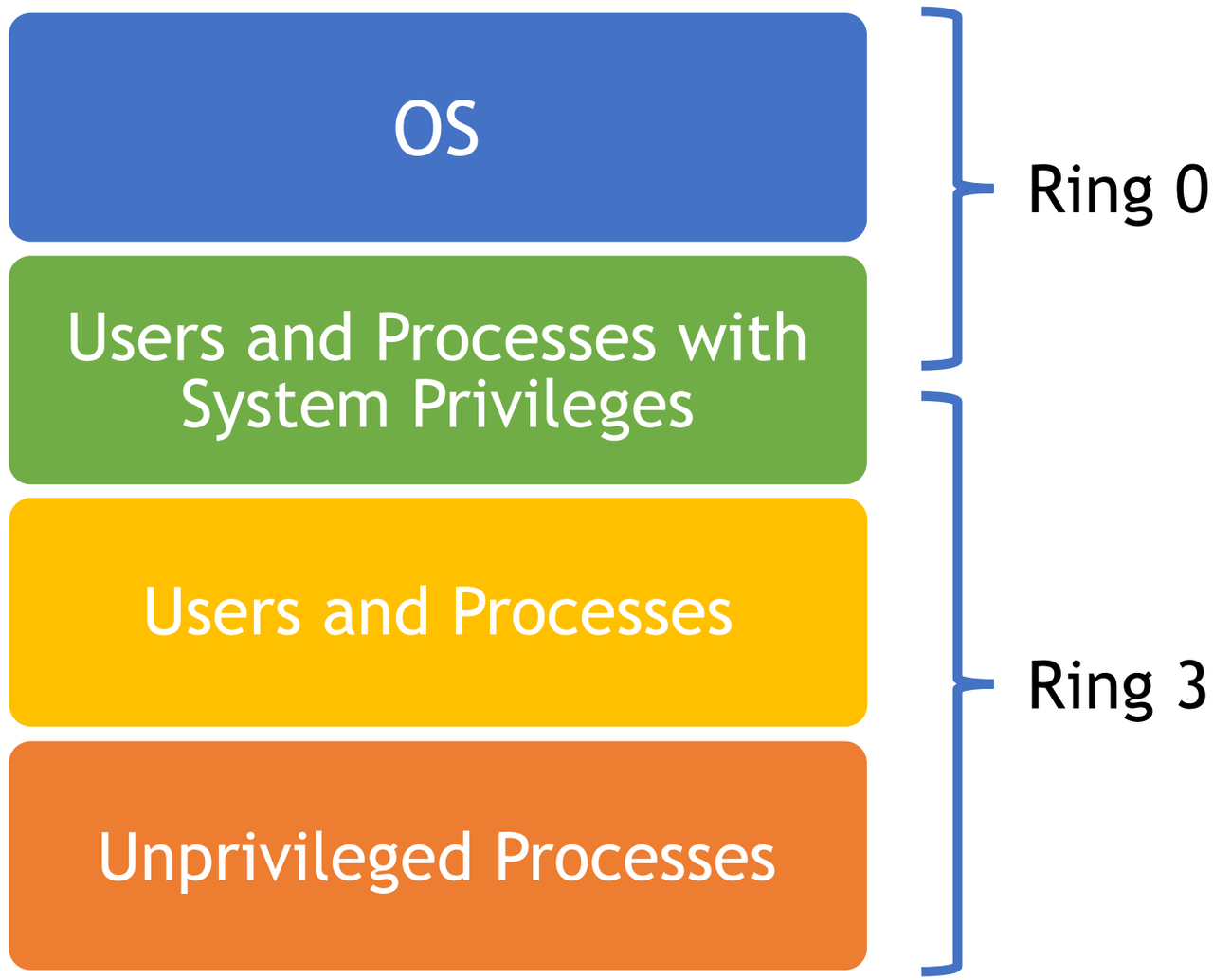
# Privilege Hierarchy

Device drivers, kernel modules, etc.

sudo, “administrator” accounts, OS services

Everything that is isolated and subject to access control

chroot jails, containers, low-integrity processes



# Example: Chrome Multiprocess Architecture

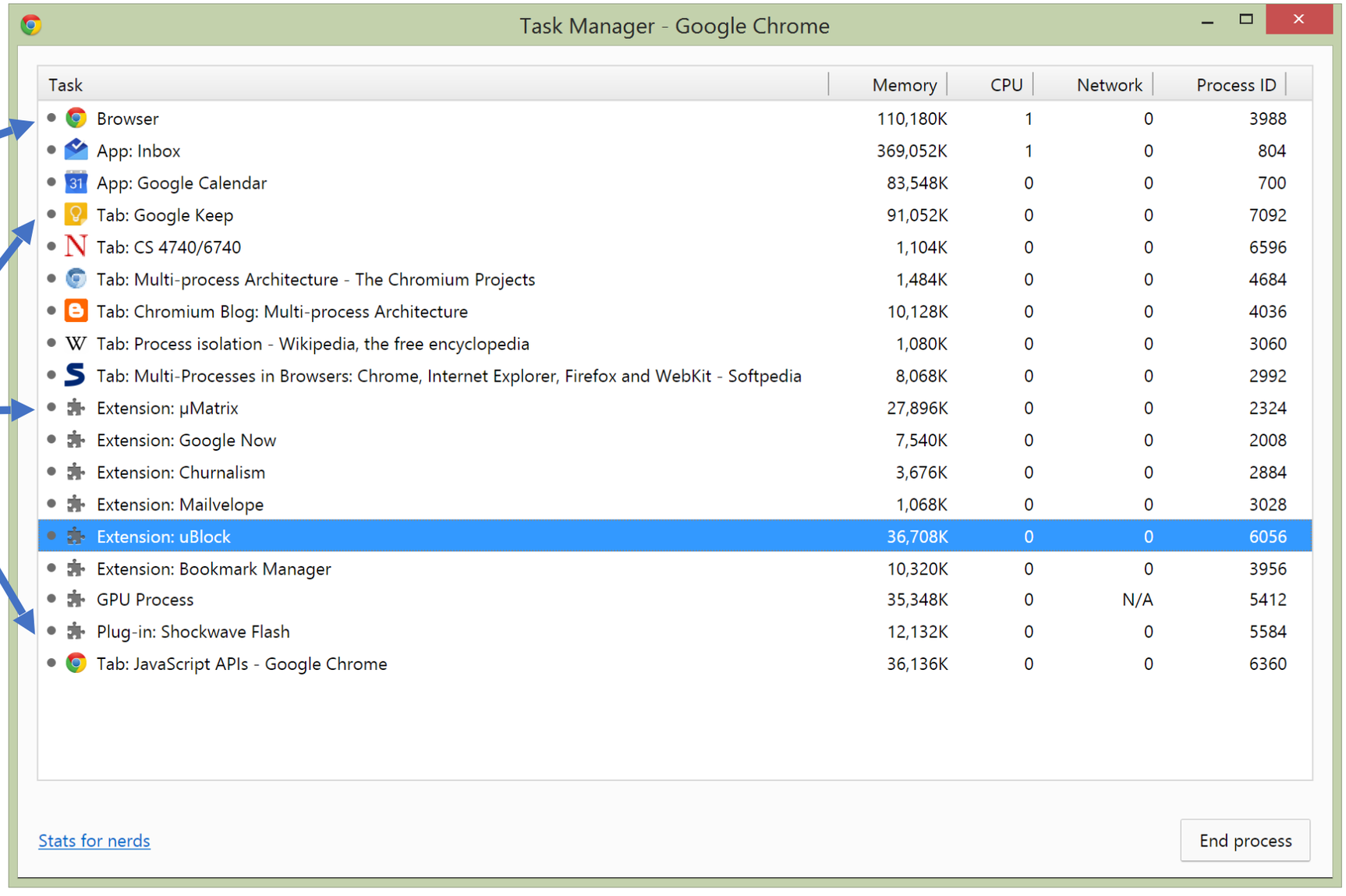
Chrome is split across many processes

"Core" process has user-level privileges

- May read/write files
- May access the network
- May render to screen

Each tab, extension, and plugin has its own process

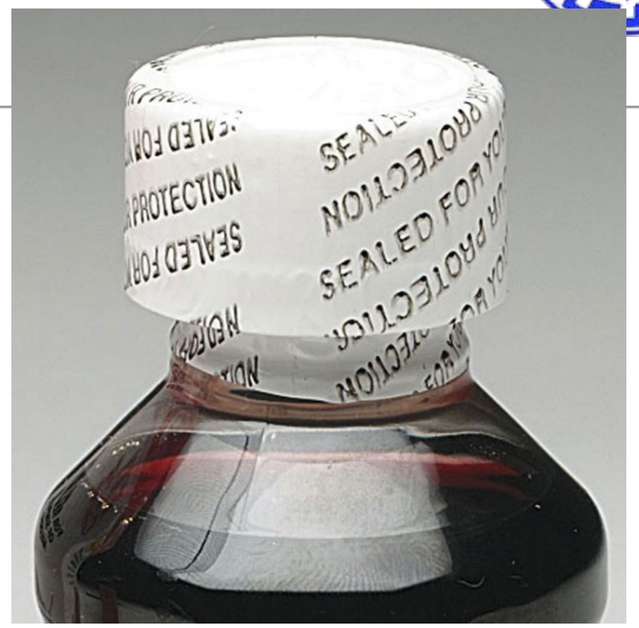
- Parse HTML, CSS, JS
- Execute JS
- Large attack surface!
- Thus, have **no privileges**
- All I/O requests are sent to the core process

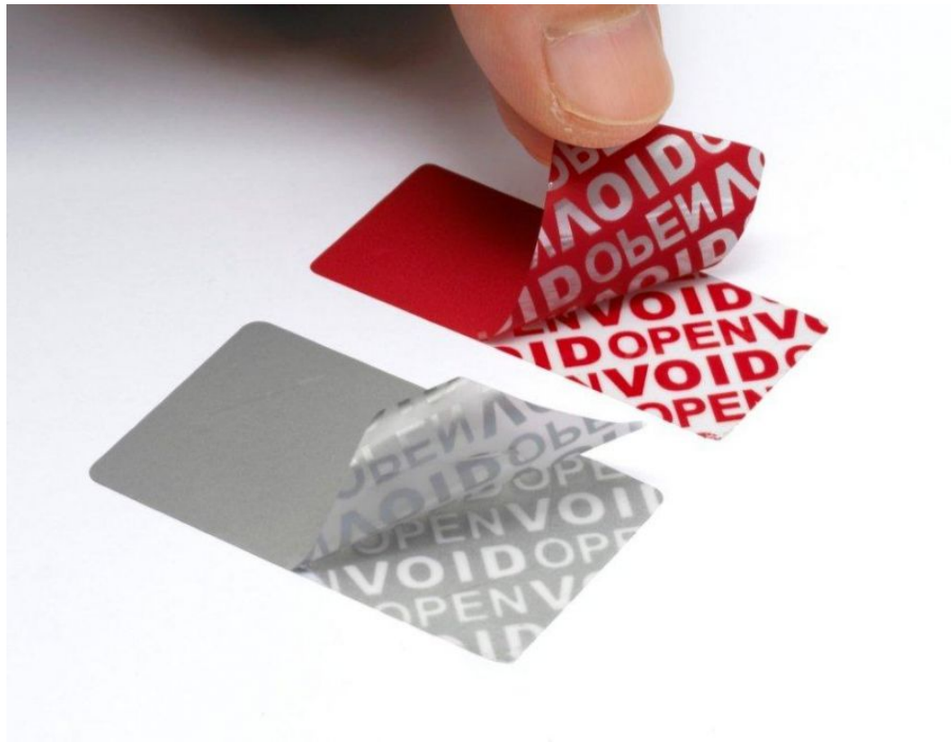


# Compromise Recording

- Concede that attacks will occur, but record the fact

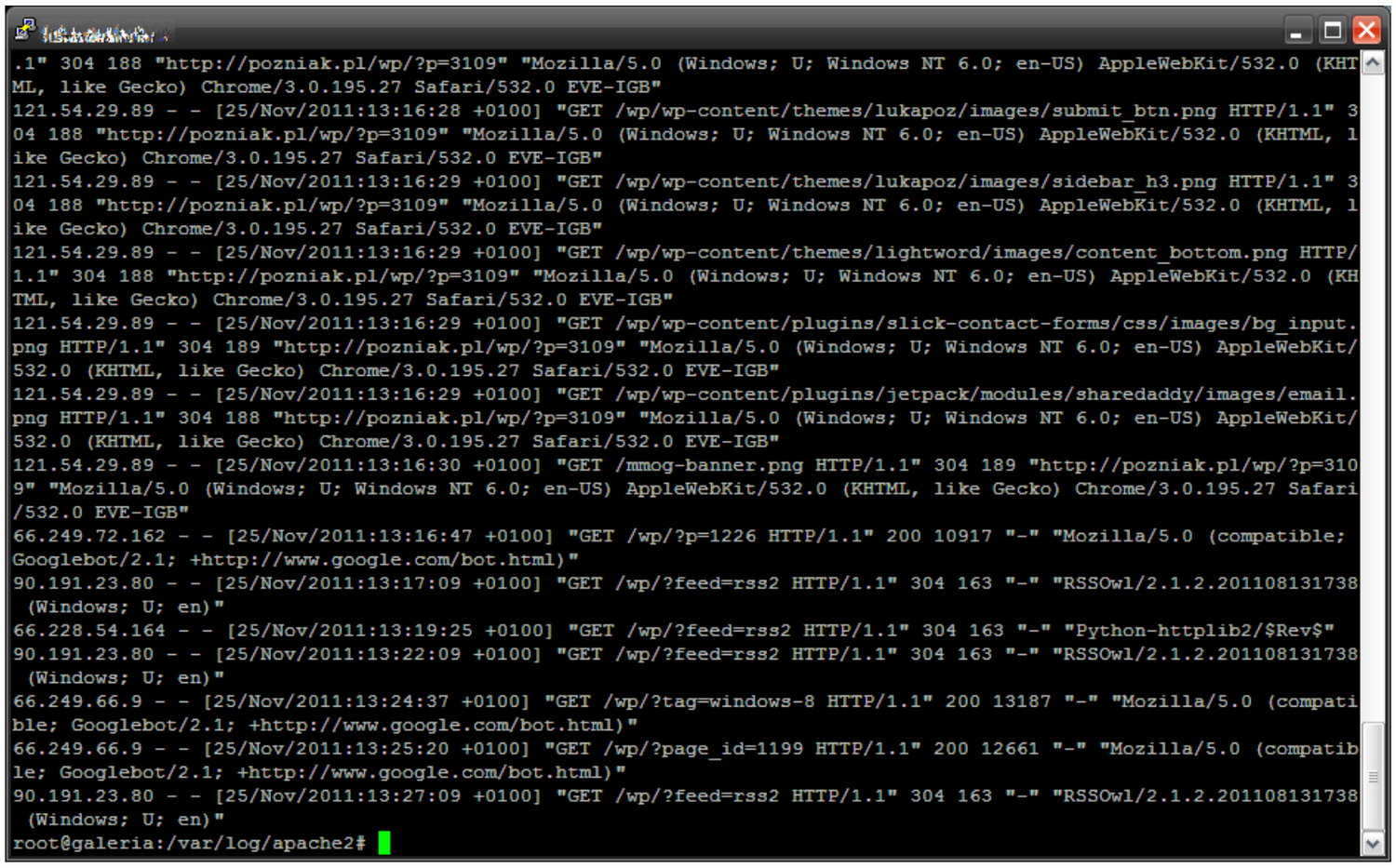
- Auditing approach to security
  - Detection and recovery
  - "Tamper-evident" vs. "Tamper-proof"



# Logging

- Log everything
- Better yet, use remote logging
  - Ensures that attacker with local access cannot erase logs
- Logs are useless if they aren't monitored
- Advanced approaches
  - Intrusion Detection Systems (IDS)
  - Anomaly detection
  - Machine learning-based approaches

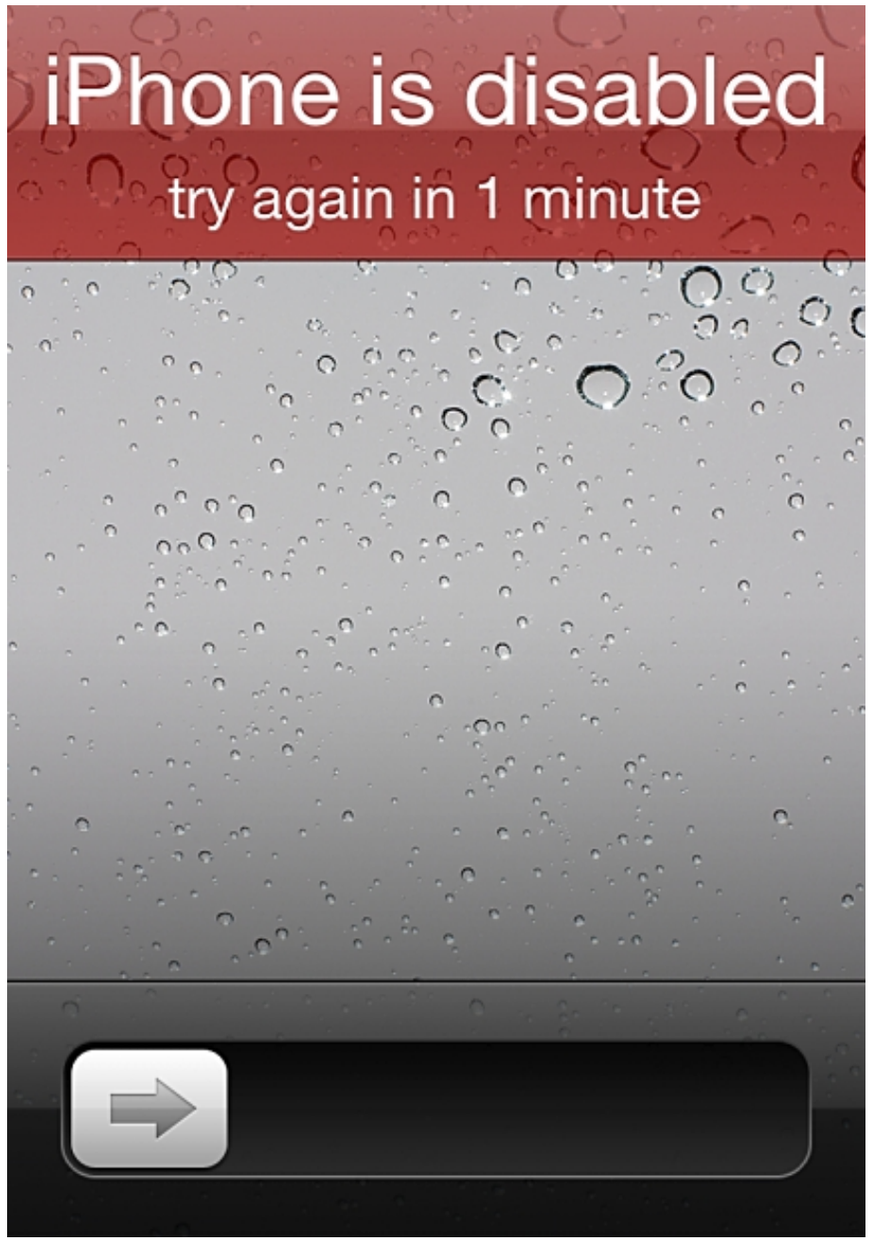


# Work Factor

- Increase the difficulty of mounting attacks
- Sometimes utilizes non-determinism
  - e.g. increasing entropy used in ASLR
- Sometimes utilizes time
  - Increase the lengths of keys
  - Wait times after failed password attempts

# Authentication Rate Limiting

- Short delay after each failed authentication attempt
  - Delays may increase as the consecutive failed attempts increase
- Does not prevent password cracking attempts, but slows them down



# Open Design

- Kerckhoff's Principle: A cryptosystem should be secure even if everything about the system, except the key, is public knowledge

- Generalization: A system should be secure even if the adversary knows everything about its design
  - Design does not include runtime parameters like secret keys
- Contrast with “security through obscurity”

# Open Design Example:

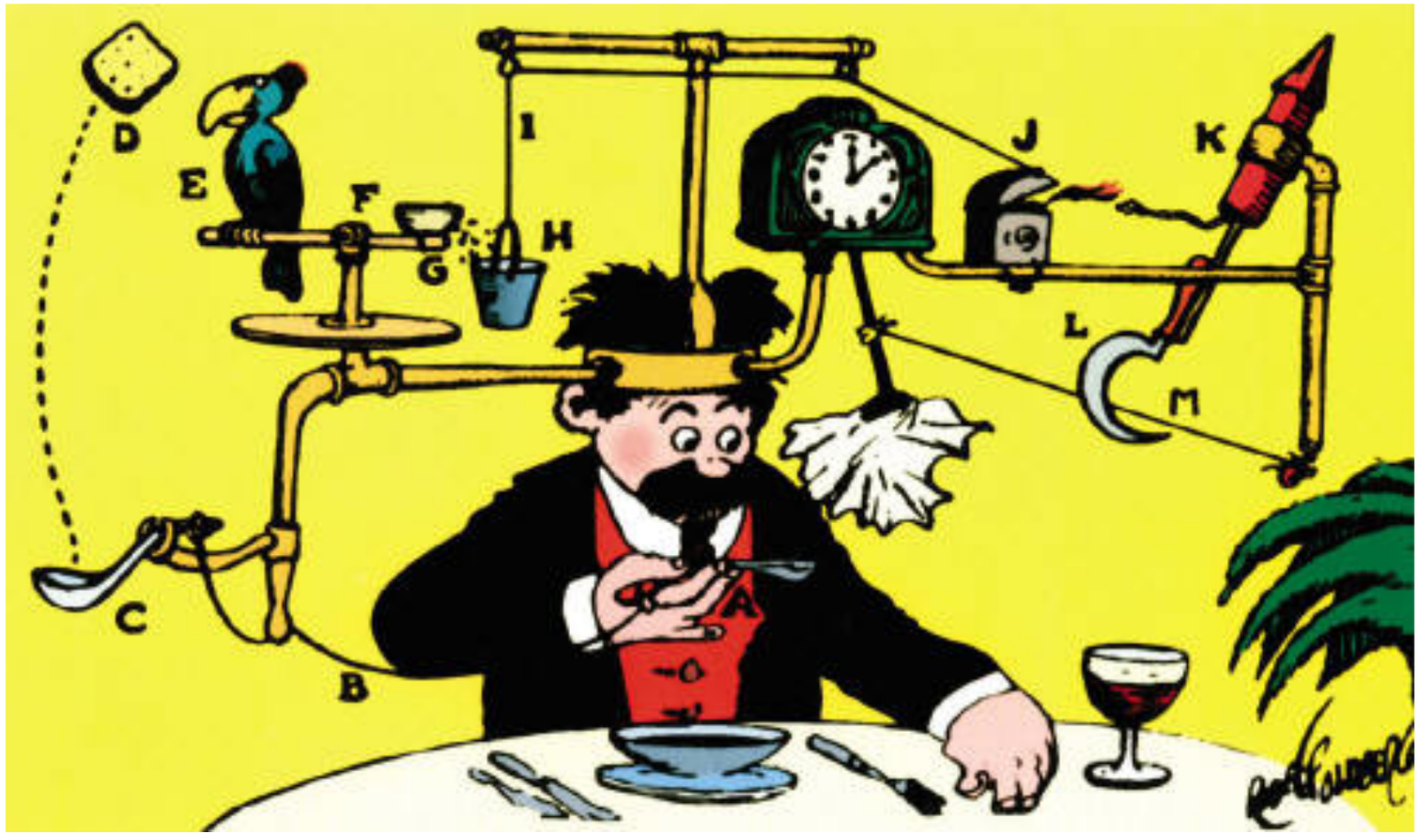
- Problem: MPAA wants control over DVDs.
  - Region coding, unskippable commercials.
- Solution: CSS (Content Scrambling System)
  - CSS algorithm kept secret.
  - DVD Players need player key to decrypt disk key on DVD to descript movie for playing.
    - Encryption uses 40-bit keys.
    - People w/o keys can copy but not play DVDs.
- What happened next?
  - CSS algorithm reverse engineered.
  - Weakness in algorithm allows disk key to be recovered in an attack of complexity 225, which takes only a few seconds.

# Closed Source

- Security through obscurity.
- Assumes code in binary can’t be read
  - what about disassemblers?
  - what about decompilers?
  - what about debuggers?
  - what about strings, lsof, truss, /proc?
- Reverse engineering.

# Economy of Mechanism

Would you depend on a defense system designed like this?



# Economy of Mechanism

- Simplicity of design implies a smaller attack surface

- Correctness of protection mechanisms is critical
  - "Who watches the watcher?"
  - We need to be able to trust our security mechanisms
    - (Or, at least quantify their efficacy)
- Essentially the KISS principle
  - Keep it simple, stupid

# Example

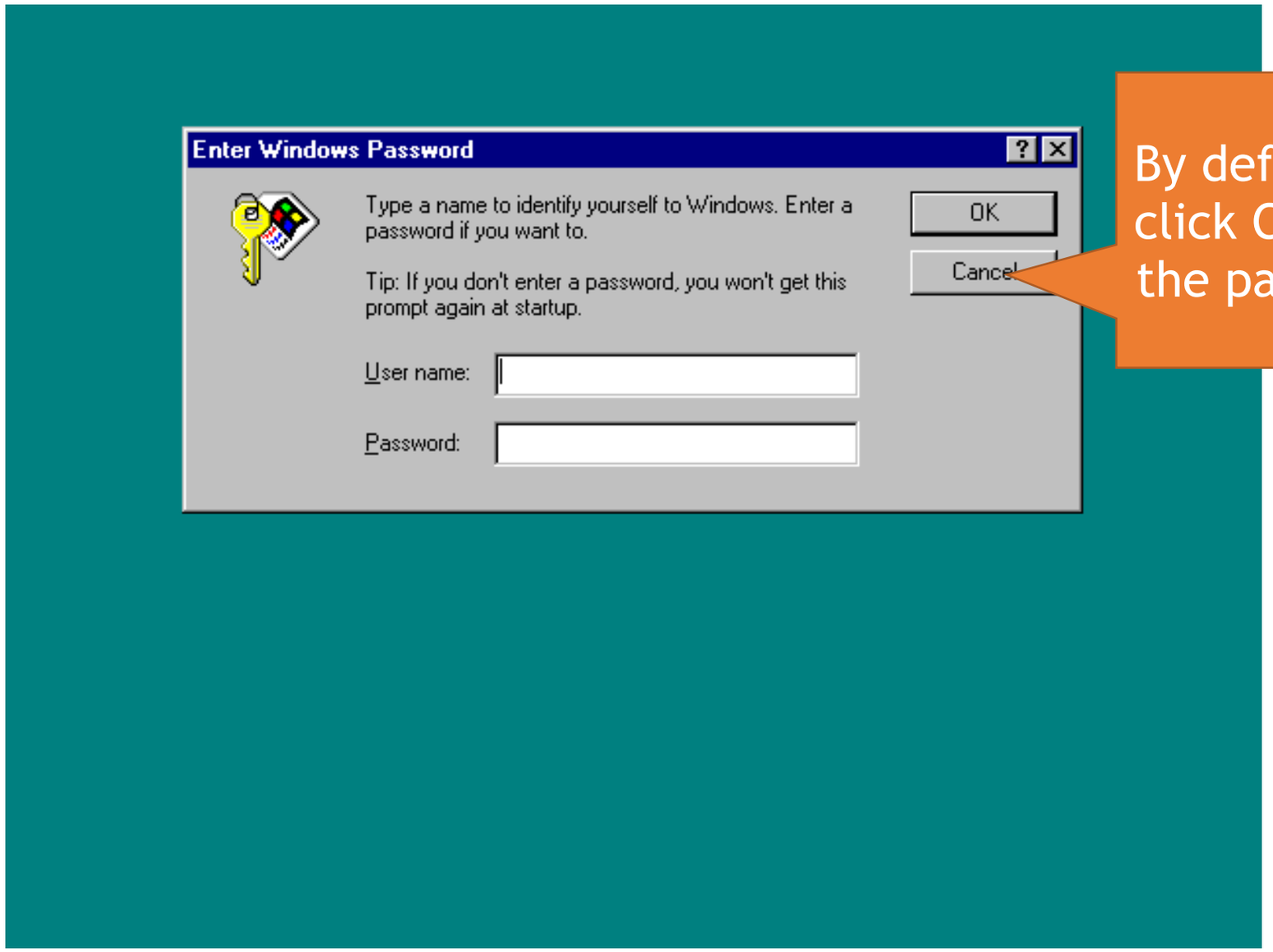
- Existing operating systems are monolithic
  - Kernel contains all critical functionality
  - Process and memory management, file systems, network stack, etc…
- Micro-kernel OS
  - Kernel only contains critical functionality
    - Direct access to hardware resources
    - Process and memory management
    - Small attack surface
  - All other functionality runs in separate processes
    - File systems, network stack, device drivers
- Examples
  - 1) GNU Hurd 2) seL4 – formally verified!

# Complete Mediation

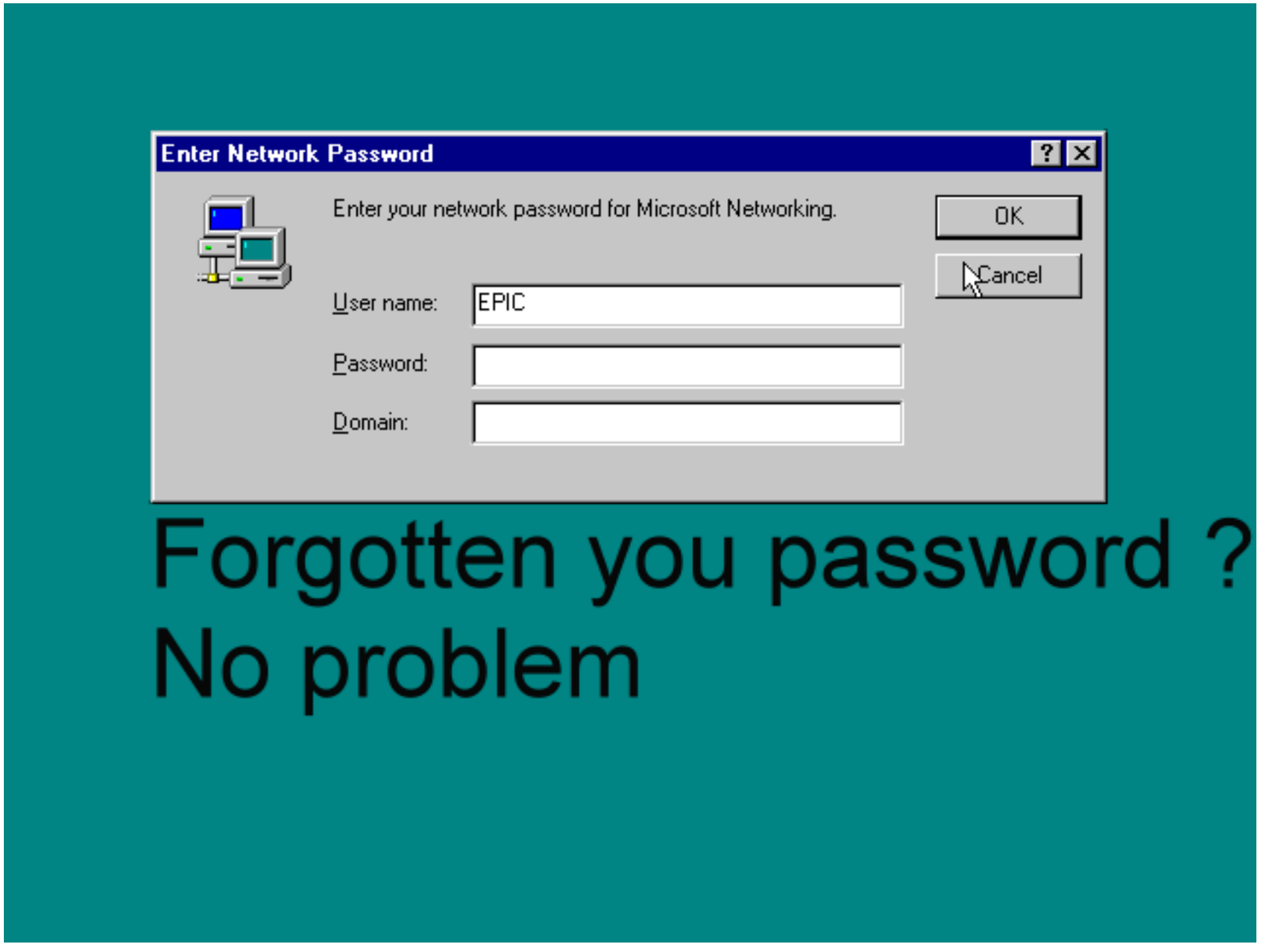
# Complete Mediation

- Every access to every object must be checked for authorization

- Incomplete mediation implies that a path exists to bypass a security mechanism
- In other words, isolation is incomplete


Enter Windows Password
Type a name to identify yourself to Windows. Enter a password if you want to.
Tip: If you don't enter a password, you won't get this prompt again at startup.
User name:
Password:
OK
Cancel
By default, user could click Cancel to bypass the password check :(

[Wilson'19]


Enter Network Password
Enter your network password for Microsoft Networking.
OK
Cancel
User name:
EPIC
Password:
Domain:
Forgotten you password ?
No problem

[Wilson'19]

# Case Studies

# Isolation

- Confinement:  ensure application does not deviate from pre-approved behavior
- Can be implemented at many levels:
  - Hardware: run application on isolated hw  (air gap)
    - difficult to manage
  - Sandboxing:
    - System call interposition (Isolates a process in a single operating system)
    - Isolating threads sharing same address space (Software Fault Isolation)
    - Application specific:  e.g.  browser-based confinement
  - Virtual machines:  isolate OS's on single hardware

# Example: Java Sandbox

# Java history

| Version | Release date | End of Free Public Updates[5][6] | Extended Support Until |
|---|---|---|---|
| JDK Beta | 1995 | ? | ? |
| JDK 1.0 | January 1996 | ? | ? |
| JDK 1.1 | February 1997 | ? | ? |
| J2SE 1.2 | December 1998 | ? | ? |
| J2SE 1.3 | May 2000 | ? | ? |
| J2SE 1.4 | February 2002 | October 2008 | February 2013 |
| J2SE 5.0 | September 2004 | November 2009 | April 2015 |
| Java SE 6 | December 2006 | April 2013 | December 2018 |
| Java SE 7 | July 2011 | April 2015 | July 2022 |
| Java SE 8 (LTS) | March 2014 | **January 2019 for Oracle (commercial)**<br>December 2020 for Oracle (personal use)<br>At least September 2023 for AdoptOpenJDK | March 2025 |
| Java SE 9 | September 2017 | March 2018 for OpenJDK | N/A |
| Java SE 10 | March 2018 | September 2018 for OpenJDK | N/A |
| Java SE 11 (LTS) | September 2018 | At least September 2022 for AdoptOpenJDK | September 2026 |
| **Java SE 12** | March 2019 | September 2019 for OpenJDK | N/A |

**Legend:** Old version | Older version, still supported | **Latest version**

[wikipedia.com]

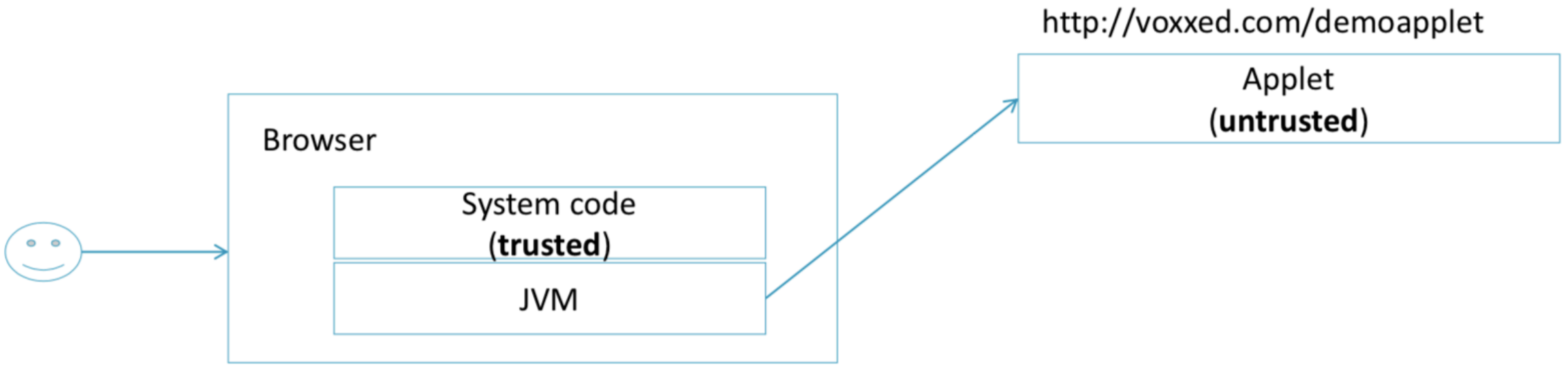
# Evolution of the Java security model

- With the introduction of various technologies for loading and executing code on the client machine from the browser (such as Applets) - a new range of concerns emerge related to client security – this is when the Java security sandbox starts to evolve ...

- The goal of the Java security sandbox is to allow untrusted code from applets to be executed in a trusted environment such as the user's browser

# Evolution of the Java security model

- JDK 1.0 (when it all started ...) – the original sandbox model was introduced




- Code executed by the JVM is divided in two domains – trusted and untrusted
- Strict restriction are applied by default on the security model of applets such as denial to read/write data from disk, connect to the network and so on

# The Security Model

- The Java Security Model is made up of three primary pieces:
  - The Bytecode Verifier
  - The Class Loader
  - The Security Manager

# The Bytecode Verifier

- Once bytecodes have been loaded in to the machine but before they are run:
  - Opcodes are checked
  - Addresses are verified to access only memory in the virtual machine
  - Strict type enforcement
- Only verified code is run on the JVM

# The Class Loader

- Imported class are each run in their own namespace
- Built-in classes are all run in a single namespace
- Class loader always searches the built-in name space for a requested class first so as to avoid running a downloaded class with the same name.
- Built-in classes are considered to be “trusted” and are always run in preference of a downloaded class of the same name.

# The Security Manager

- Each application can have an individual security policy
- Security policies are defined in external files that are accessible by the security manager
- The security manager enforces the specified security policy
- The application security is made up of two pieces:
  - A system piece, found in java.home\lib\security
  - An application specific piece in user.home\lib\security (or anywhere you want to put it)

# Policy Files

grant [signedBy “signer_names”,] [codebase “URL”] {

permission permission_class_name

“target_name”, [“action”]

[, signedBy “signer_names”];

Ex.

grant signedBy “ACME Software” codebase http://www.acme.com/-

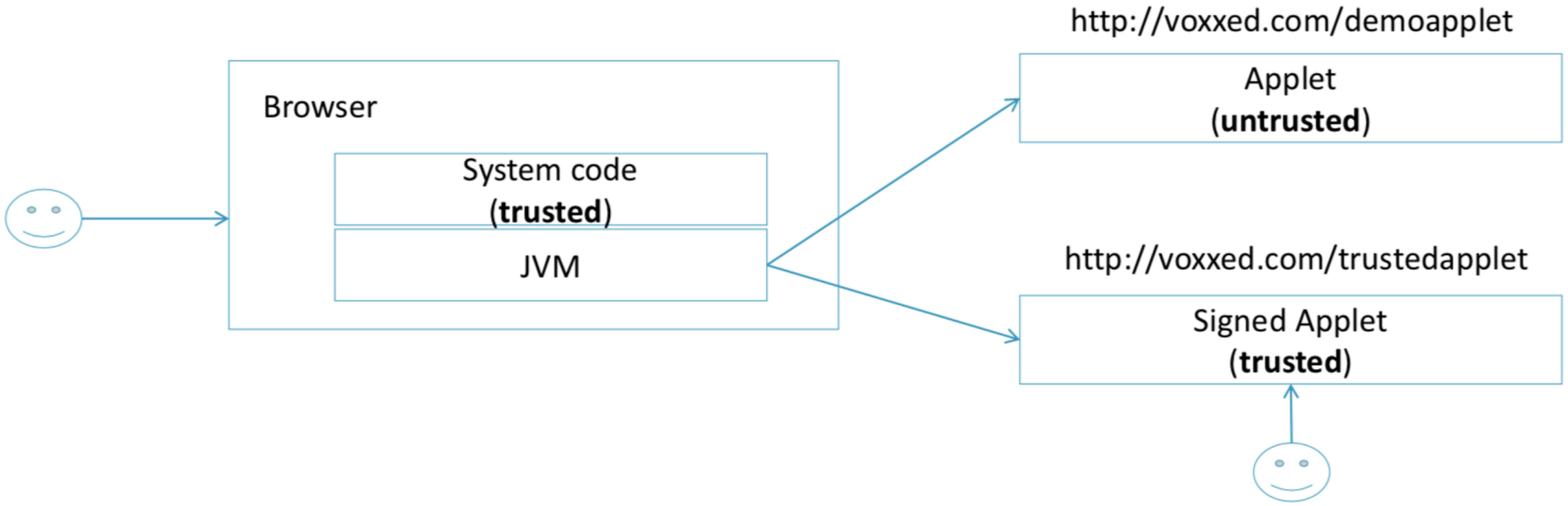
{ permission java.io.FilePermission “c:\\autoexec.bat”,”read”;

permission java.lang.RunTimePermission “queuePrintJob”;

}

# Evolution of the Java security model

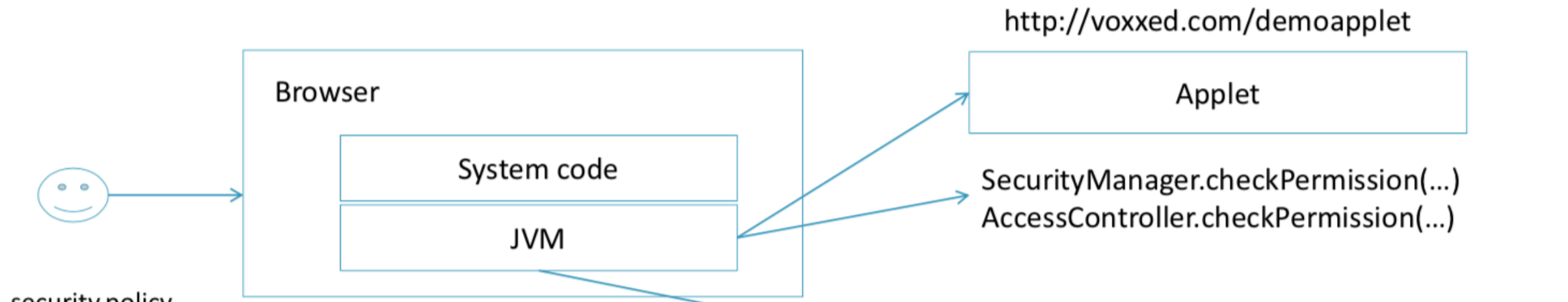
- JDK 1.1 (gaining trust ...) – applet signing introduced




- Local code (as in JDK 1.0) and signed applet code (as of JDK 1.1) are trusted
- Unsigned remote code (as in JDK 1.0) is not trusted

# Evolution of the Java security model

- JDK 1.2 (gaining more trust ...) – fine-grained access control




security.policy

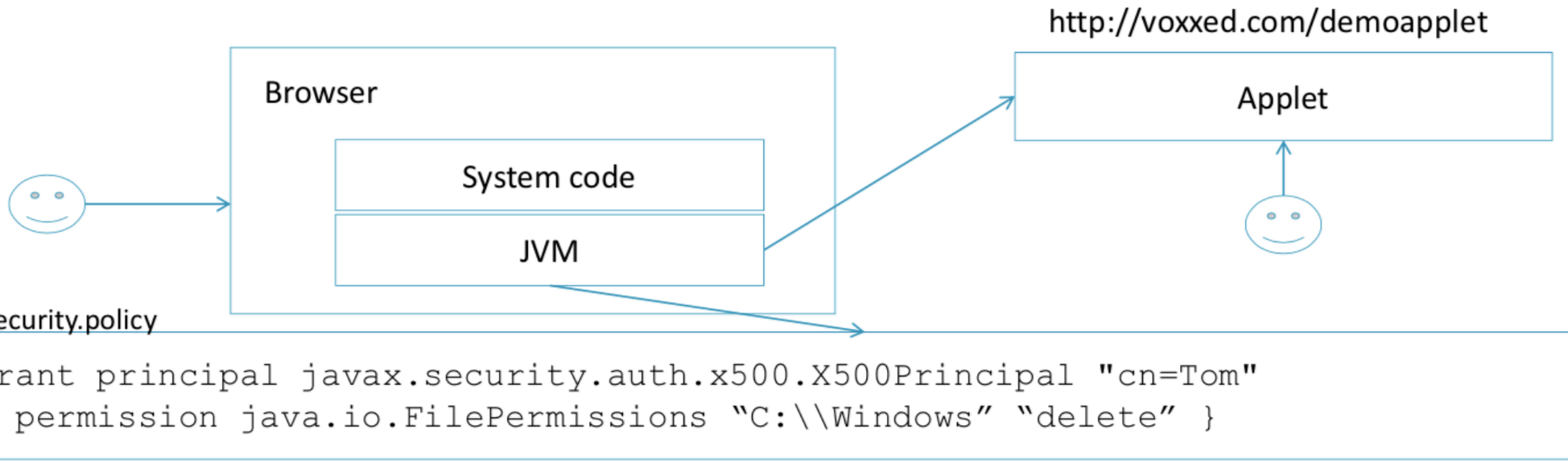
```
grant codeBase http://voxxed.com/demoapplet {
        permission java.io.FilePermisions “C:\\Windows” “delete”
}
```

# Evolution of the Java security model

- The security model becomes code-centric
  - Permissions depend on where the source code is loaded from
- Additional access control decisions are specified in a security policy
- No more notion of trusted and untrusted code
- The notion of protection domain introduced – determined by the security policy
- Two types of protection domains – system and application

# Evolution of the Java security model

- JDK 1.3, 1,4 (what about entities running the code ... ?) – JAAS



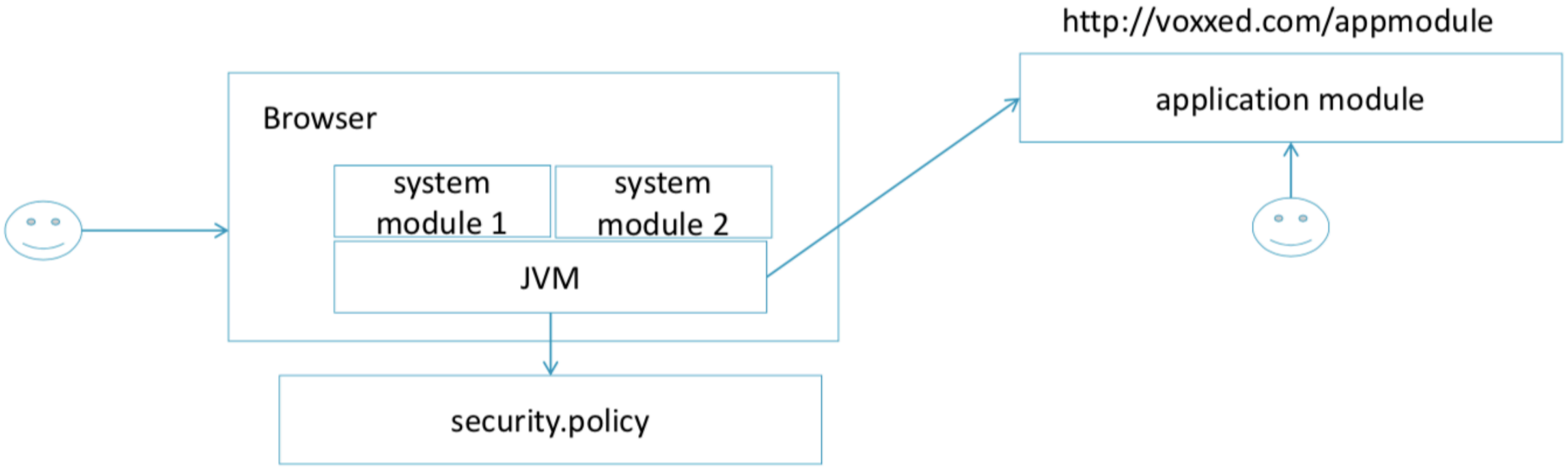

- JAAS (Java Authentication and Authorization Service) extends the security model with role-based permissions
- The protection domain of a class now may contain not only the code source and the permissions but a list of principals

# Evolution of the Java security model

- JDK 1.5, 1.6 (enhancing the model ...) – new additions to the sandbox model (e.g. LDAP support for JAAS)
- JDK 1.7, 1.8 (further enhancing the model ...) – enhancements to the sandbox model (i.e finer grained permissions)

# Evolution of the Java security model

- JDK 1.9 and beyond ... (applying the model to modules ...)



# Acknowledgments/References


- [Wilson'19] CS 2550 - Foundations of Cybersecurity, Christo Wilson, Northeastern University, Spring 2019
- [Garfinkel'04] CSCI E-170: Computer Security, Usability & Privacy, Simson L. Garfinkel, MIT, 2004
- [Walden'12] CSC 666 -- Secure Software Engineering, James Walden, Northern Kentucky University, Fall 2012
- [Boneh'15] CS 155, Computer Security, Dan Boneh, Stanford University, 2015
- [Steflik'13] CS-328 Internet and Mobile Programming, Dick Steflik, Binghamton University, Fall 2013
- [Toshev'16] Security architecture of the Java platform, Martin Toshev, Voxxed Days Luxembourg, 2016