Lecture 17: Security

Professor G. Sandoval

Some slides derived from : Tanenbaum/Bo, and Brendan Dolan-Gavitt
Thanks !!

Security

- → The Security Environment
- OS Security
- Controlling Access to Resources
- Formal Models of Secure Systems
- Basics of Cryptography

Computer Security

- Security is a big field, encompassing:
  - Application security
  - Network security
  - Authentication
  - Digital forensics
  - Cryptography
  - Privacy/Anonymity

Operating Systems Security

- In this course we will only worry about security as it applies to operating systems
- This is some mix of *authentication*, *access control*, and *application security*

Computer Security

- Generally, we talk about computer security in three broad categories:
  - **Confidentiality** – Exposure of data or preventing others from finding out information we don't want them to have
  - **Integrity** – Tampering with Data or preventing others from modifying our data without permission
  - **Availability** – Denial of Service or preventing others from denying us access to some service ()

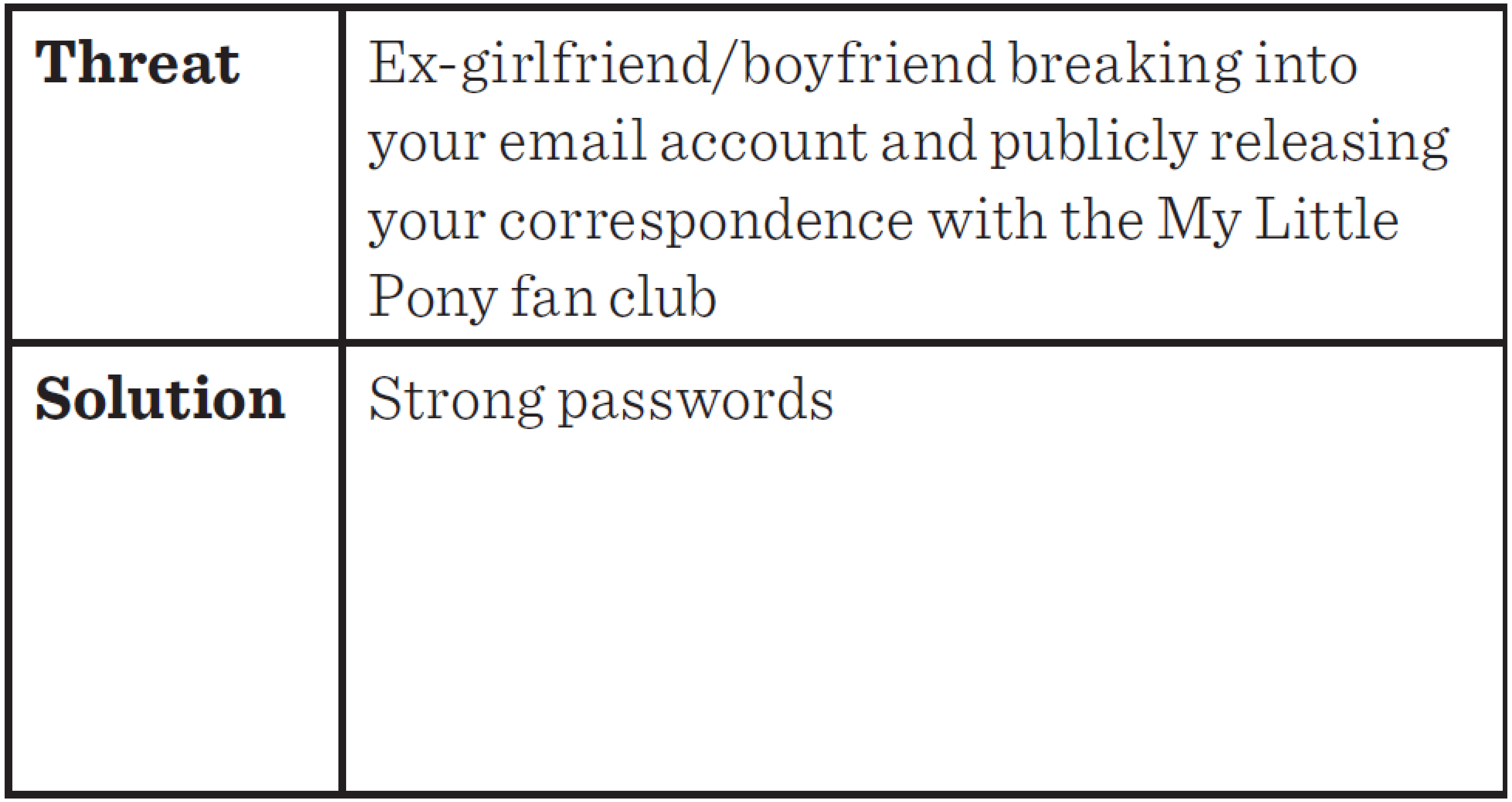
Computer Security

- Generally, we talk about computer security in three broad categories:
  - **Confidentiality** – Exposure of data or preventing others from finding out information we don't want them to have
  - **Integrity** – Tampering with Data or preventing others from modifying our data without permission
  - **Availability** – Denial of Service or preventing others from denying us access to some service ()

Threat Modeling

- It usually doesn't make sense to talk about a system being "secure" or "insecure"
- Instead, we need to be more precise:
    - What are we trying to protect?
    - Who do we need to protect against? What are their capabilities?

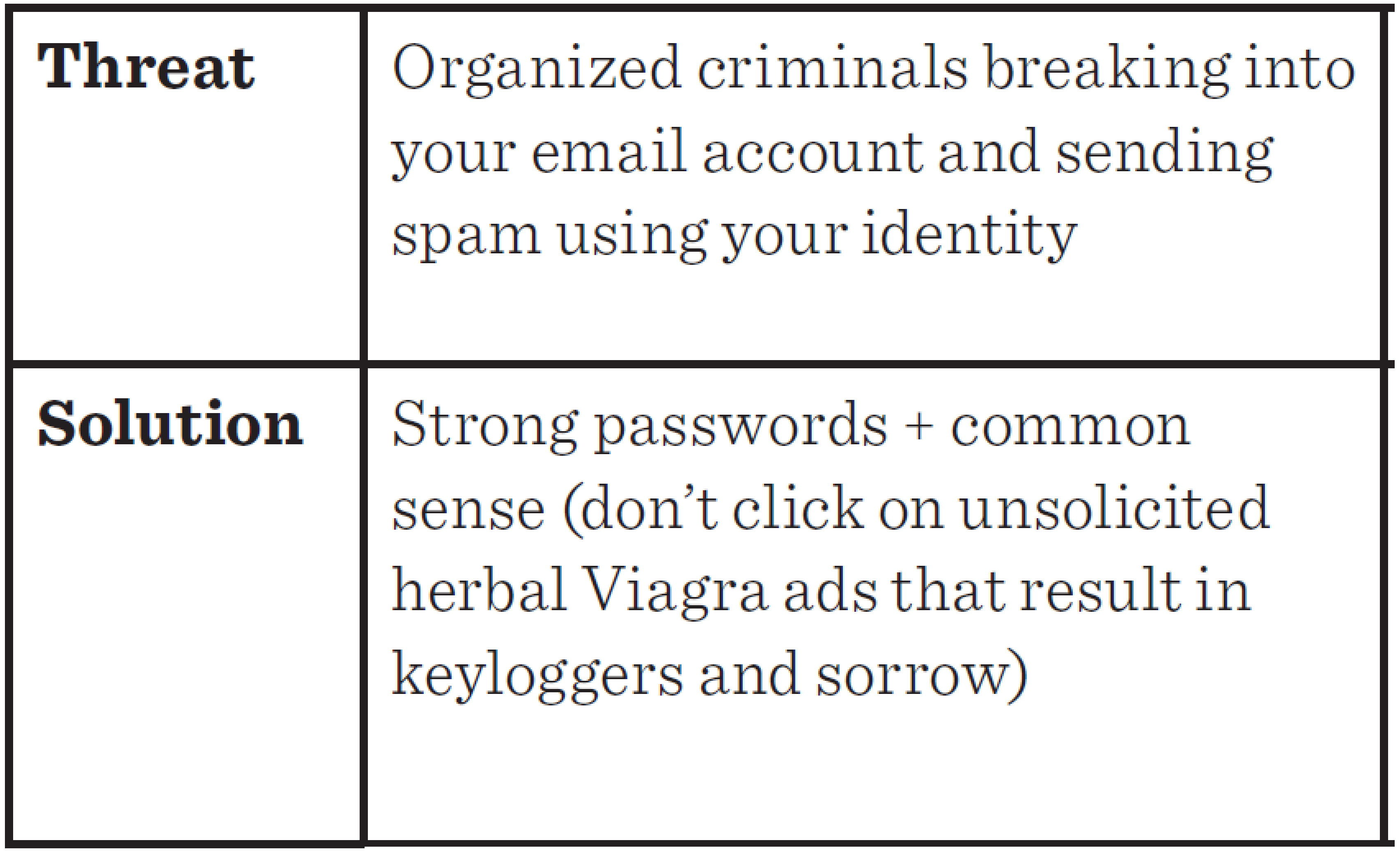
A Practical Threat Model



Threat	Ex-girlfriend/boyfriend breaking into your email account and publicly releasing your correspondence with the My Little Pony fan club
Solution	Strong passwords

Source: Mickens, *This World of Ours*

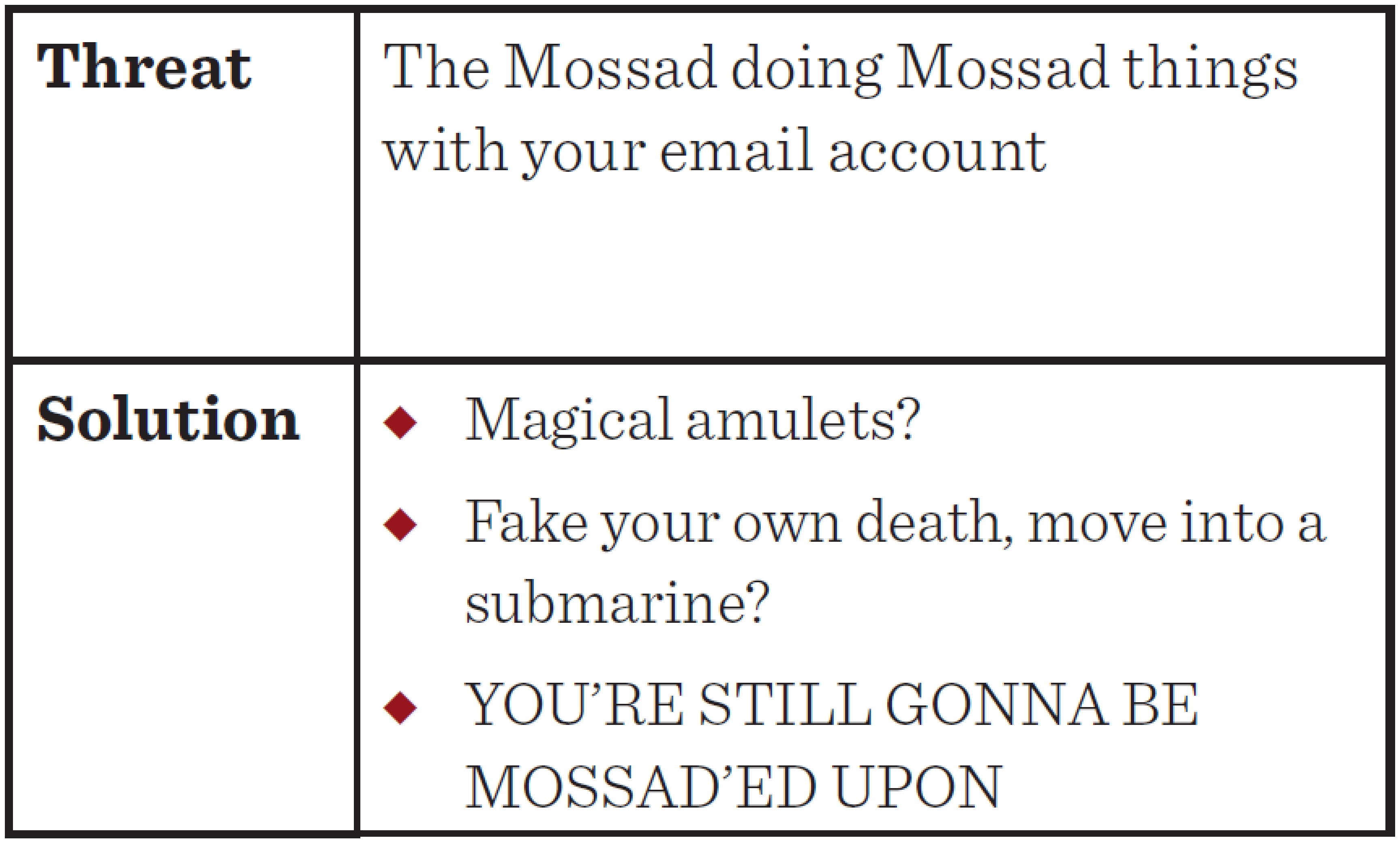
A Practical Threat Model



Threat	Organized criminals breaking into your email account and sending spam using your identity
Solution	Strong passwords + common sense (don't click on unsolicited herbal Viagra ads that result in keyloggers and sorrow)

Source: Mickens, *This World of Ours*

A Practical Threat Model



Threat	The Mossad doing Mossad things with your email account
Solution	◆ Magical amulets? ◆ Fake your own death, move into a submarine? ◆ YOU'RE STILL GONNA BE MOSSAD'ED UPON

Source: Mickens, *This World of Ours*

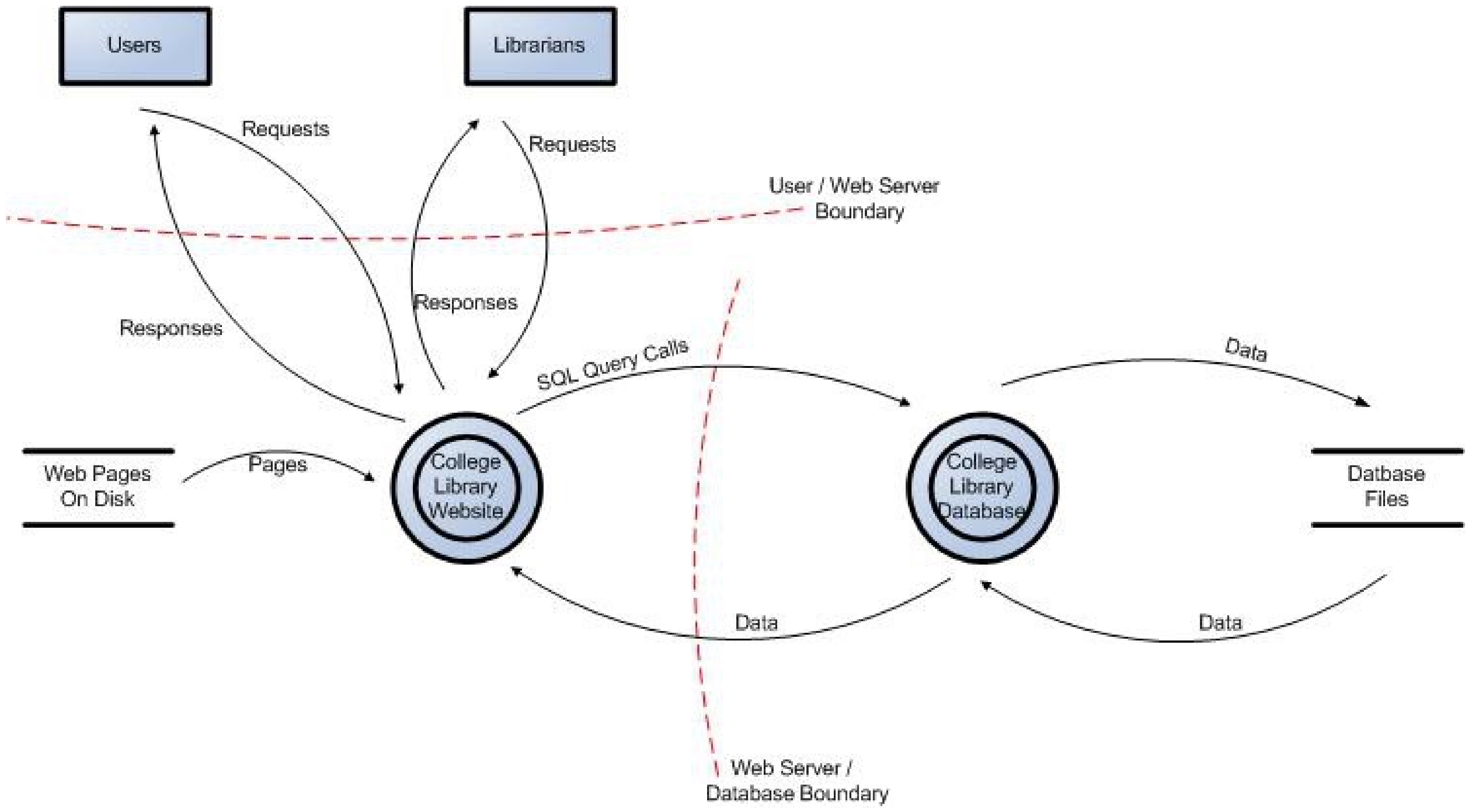
Threat Modeling

- Roughly, we can divide this into three steps:
  - System understanding
  - Threat categorization
  - Countermeasures and mitigation

System Understanding

- To properly protect a system, you need to understand it:
  - Identify assets that need protecting
  - Look at ways the system can be used and assets can be accessed
  - Figure out what rights should be granted to what assets and classes of user
  - Identify ***privilege boundaries*** – places where a program or user changes their privilege level

Example System Diagram



Source: OWASP

Threat Categorization

- Look at the system from an attacker's point of view
- What goals might an attacker have?
- How could they achieve these goals?
- May help to use a threat categorization such as **STRIDE**: **S**poofing, **T**ampering, **R**epudiation, **I**nformation Disclosure, **D**enial of Service, **E**levation of Privilege

Mitigation

- For each of the threats to some asset, come up with a plan for mitigating or nullifying the threat, for example:
  - Attacker might guess someone's password => enforce password complexity requirements
  - Attacker might snoop on network traffic => encrypt data that is sent on the network

Security

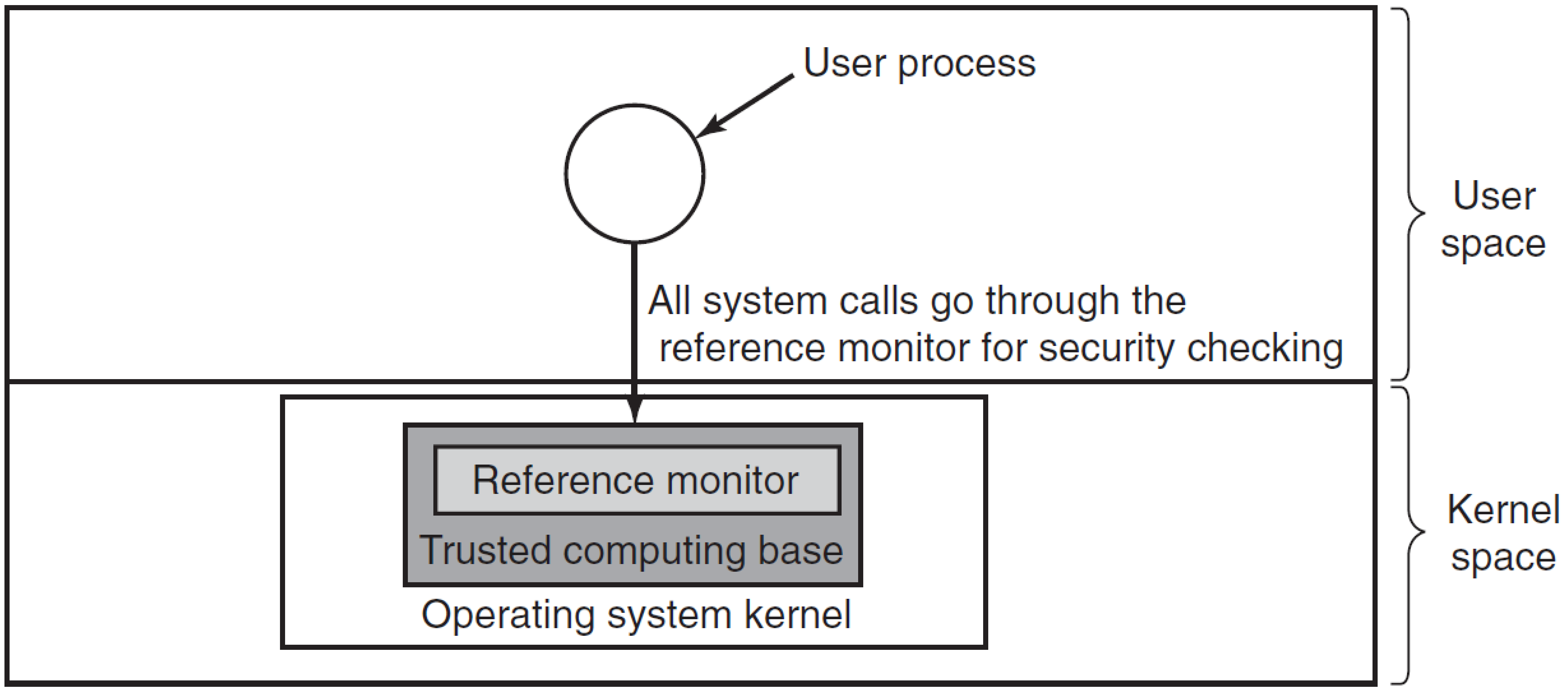
- The Security Environment
- OS Security
- Controlling Access to Resources
- Formal Models of Secure Systems
- Basics of Cryptography

Trusted Computing Base

- One strategy for building secure operating systems is to organize them into *trusted* and *untrusted* components

- The goal is that if the *trusted* component performs according to its specification, then some specific set of guarantees about security must hold

- A *reference monitor* checks all accesses between the trusted and untrusted components


User process
All system calls go through the reference monitor for security checking
User space
Reference monitor
Trusted computing base
Operating system kernel
Kernel space

Aside: Bugs & Program Size

- One rule of thumb is that as program size increases, the number of bugs increases as well
- It's harder to reason about code the more of it there is
- Therefore, if we want to be confident in our trusted computing base, we should work to *minimize* the amount of code in it
- How big is the TCB(Trusted Computed Base) for widely-used operating systems?

Security

- The Security Environment
- OS Security
- Controlling Access to Resources
- Formal Models of Secure Systems
- Basics of Cryptography

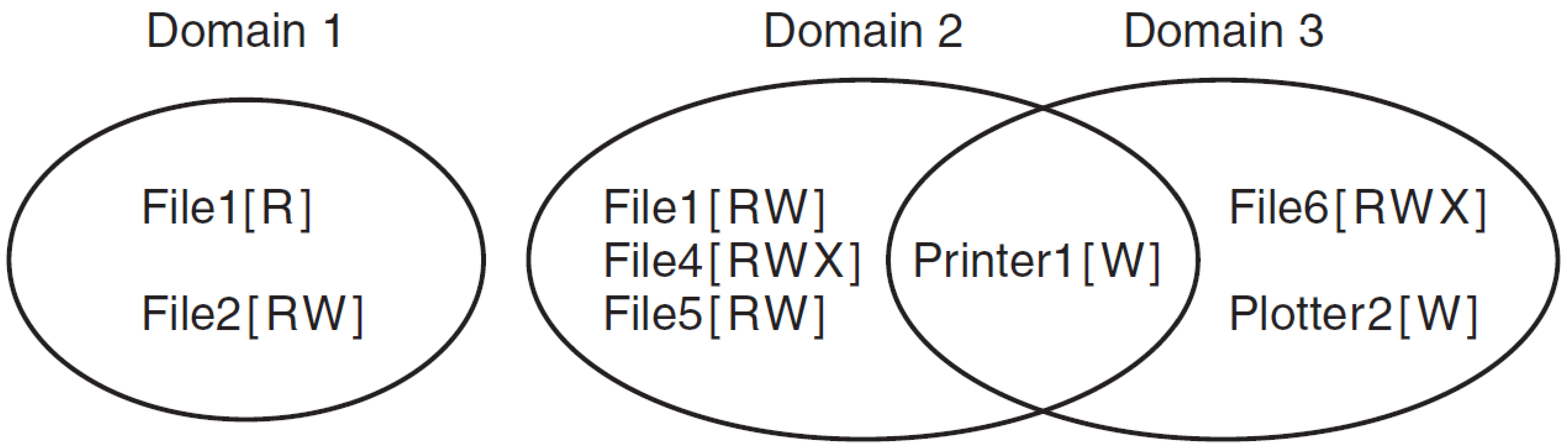
Access Control: Resources

- Systems tend to have resources that need to be protected
  - Hardware: CPUs, memory, disk drives, ...
  - Software: processes, files, databases, ...
- We'll refer to these generally as "objects" for now

Protection Domain

- A **protection domain** is a set of (object, rights) pairs
- A **right**, in this context, means an operation that can be performed on an object
- So, for example, a protection domain might correspond to a user, and the set of objects they have access to
- Or, a *group* of users that all share the same rights

Protection Domain



Principle of Least Privilege

- One principle for designing secure systems is *principle of least privilege*, also called the *principle of least authority*
- Idea: Give the minimal set of rights to an object.
- Seems obvious, but is often violated in practice

Processes and Protection Domains

- At any given time, a process operates in one protection domain
    - I.e., there is some specific set of objects that it has permission to perform some actions on
- Processes can also typically switch between protection domains as they run
    - The rules for when and how they do this vary widely between different operating systems

The UNIX Protection Model

- The protection domain of a process in UNIX is defined by its **user id (UID)** and **group id (GID)**
- The objects are files (including special files like hardware devices)

The UNIX Protection Model

- Each process is further divided into two halves: kernel mode and user mode. When the process does a **system call,** it switches from the user part to the kernel part.
- The kernel half can access a different set of objects from the user half, so the change from user to kernel is a **domain** switch
- Executable files can also have SETUID and SETGID bits, meaning that when they are executed they will run under different permissions

Protection Matrix

- How does the system keep track of which object belongs to which domain?
- We can describe domains using a *protection matrix*
- Rows represent the domains
- Columns represent the objects
- Entries in the matrix list the allowed operations

Protection Matrix

Domain	Object File1	File2	File3	File4	File5	File6	Printer1	Plotter2
1	Read	Read Write						
2			Read	Read Write Execute	Read Write		Write	
3						Read Write Execute	Write	Write

Modeling Domain Switches

- We can also use the protection matrix to model a **domain switch**
- Domains = just another object
- The associated right is the ability *enter* the domain

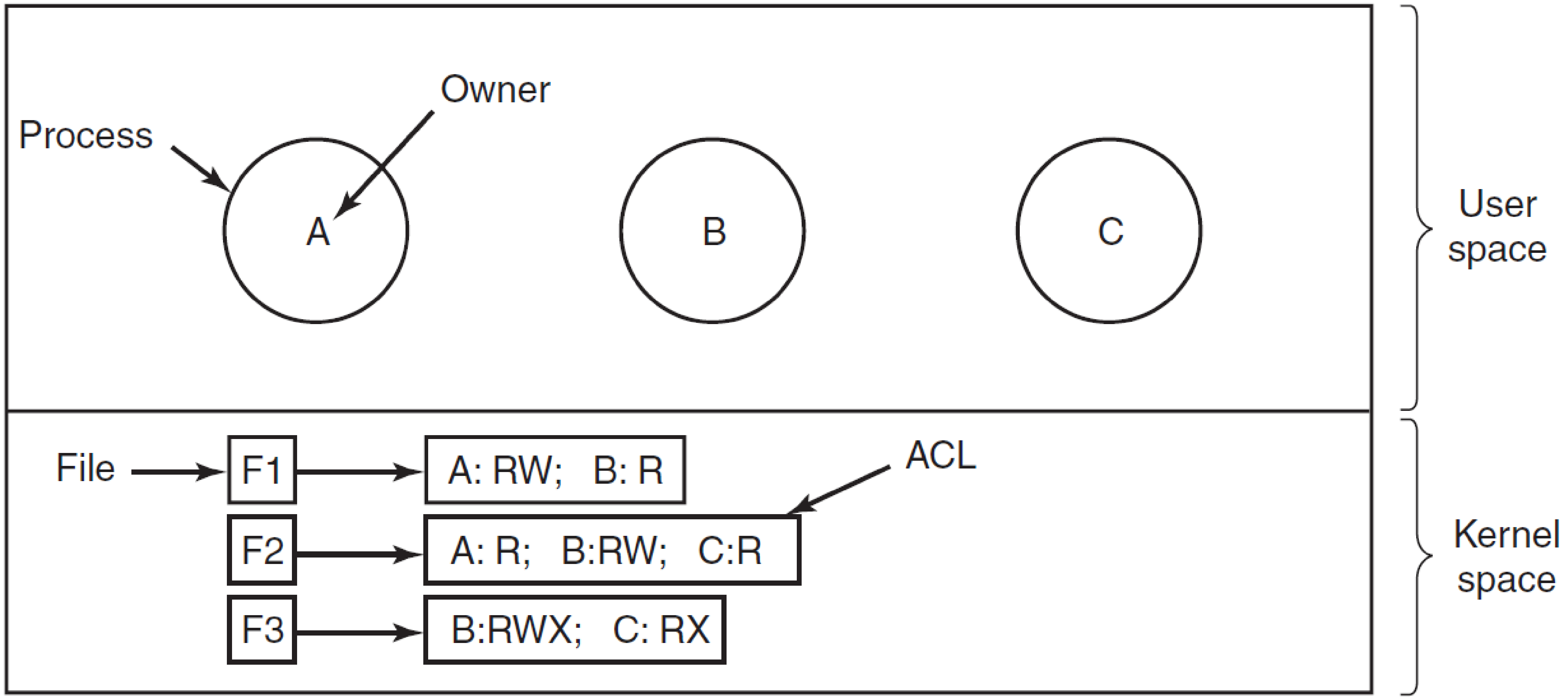
Domain Switch Matrix

omain	File1	File2	File3	File4	File5	File6	Printer1	Plotter2	Domain1	Domain2	Domain3
1	Read	Read Write								Enter	
2			Read	Read Write Execute	Read Write		Write				
3						Read Write Execute	Write	Write			

Access Control Lists

- Very few systems use a literal protection matrix to represent permissions
  - Protection matrix is large and **sparse** – wastes space
- Instead, attach to each object a list of domains and the rights for each
- This is an **access control list (ACL)**
- When talking about ACLs, we often refer to domains as "**subjects**" or "**principals**"

Basic ACL


Owner
Process
A
B
C
User space
File
F1
A: RW; B: R
ACL
F2
A: R; B:RW; C:R
F3
B:RWX; C: RX
Kernel space

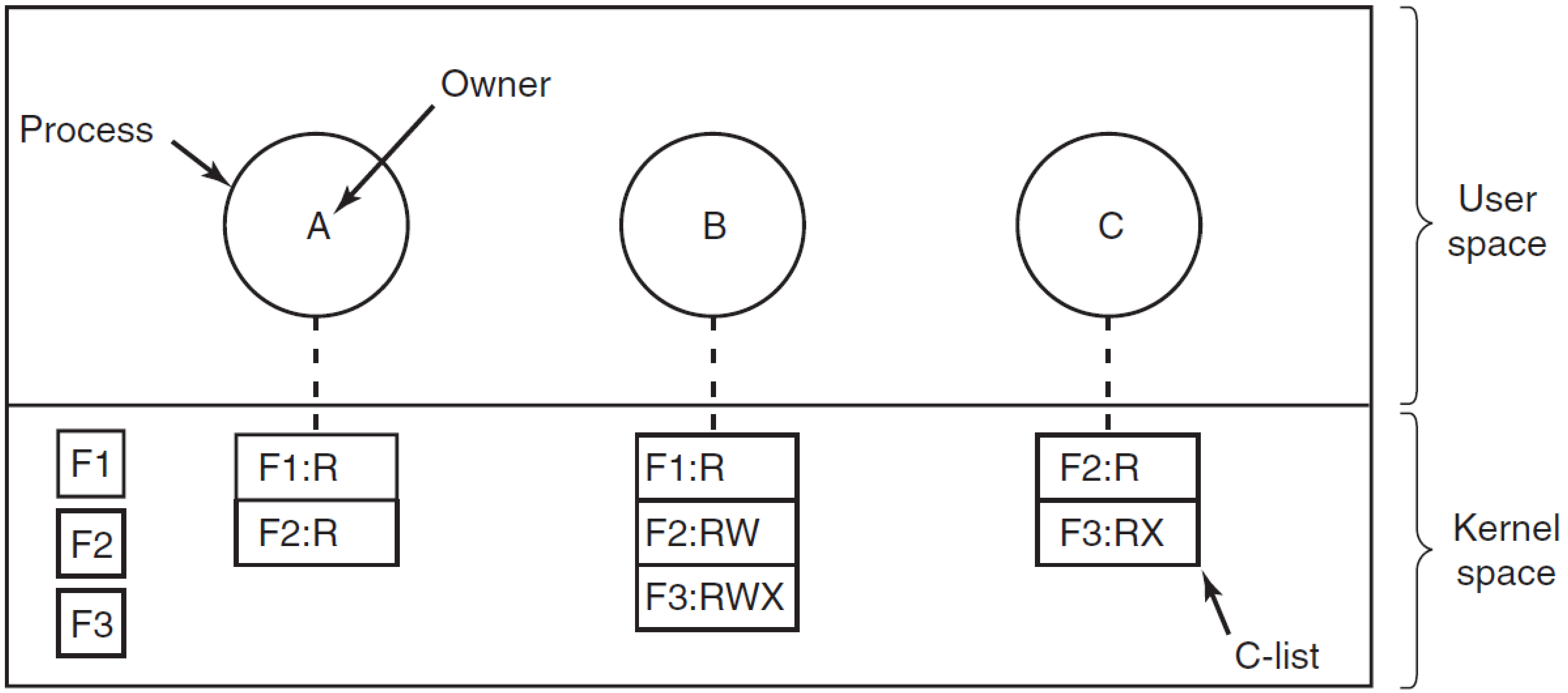
Groups and Roles

- In addition to specifying access by user, we can have *groups* of users
- Access can then be determined by what group you're currently logged in as (what **role** you currently have)
- Or, access can be granted if *any* of the groups you belong to has access (e.g., groups in UNIX)

Capabilities

- We can also **track access control** by keeping, for each process, a list of objects it can access
- This is called a **capability list (C-List)** and the individual entries are **capabilities**

Capability Lists



Representing C-Lists

- To be secure, the C-List should not be modifiable by the process itself
    - Otherwise a process could give itself arbitrary capabilities
- Three general ways to enforce this:
    - Hardware
    - Operating System: Capability descriptor
    - User Space: Cryptography

Hardware-Assisted Capabilities

- If you have a *tagged architecture*, each word in memory can have a bit associated with it (a **tag**)
- If a word is tagged, then it refers to a capability and can only be modified in kernel mode
- IBM AS/400 is one example of hardware that supported this

Capability Descriptors

- If you don't have hardware support, you can also move the C-Lists into the kernel
- Now, to access an object, you refer to a *descriptor* and ask the kernel to carry out the operation
- The descriptor is often just the index into the table of capabilities stored in the kernel
- The Hydra operating system (1974) used this technique

Cryptographic Capabilities

- Cryptography provides us with a way to store a capability wherever we like (even in user space) while protecting it from modification

- Suppose we have a **cryptographic one-way function**:
  f(Data, Secret) => Output (fixed length)

- Where it is computationally infeasible to figure out Secret if we only have Output

Cryptographic Capabilities

- To give a process a capability, kernel gives out
  ObjectId, Rights, Token
  where Token = f(ObjectId+Rights, Secret)
- Now when the process wants to access something, it includes the capability, including the token
- The kernel can now compute f(ObjectId+Rights, Secret) and compare it to the Token
- If they match, we know the ObjectId and Rights have not been modified

Cryptographic Capabilities

- The nice thing about using cryptography in this way is that we can store the capability *anywhere*
- In particular, capabilities can even be given out to remote machines that we don't control
- This makes them ideal for *distributed systems*
- Example: the Kerberos protocol uses capabilities this way (though it calls them *tickets*)

DAC and MAC

- Most OSes use a policy of **discretionary access control**: users get control over who has access to their files

- This isn't appropriate for every scenario, though: there may be policies that mandate that some kinds of information cannot be shared

- In this case we need **mandatory access control**

Security

- The Security Environment
- OS Security
- Controlling Access to Resources
- Formal Models of Secure Systems
- Basics of Cryptography

Multi-Level Security

- The usual way to do mandatory access control is by splitting users and information into access levels
- Classic example: unclassified, secret, top secret
- However, there still has to be some policy for allowing information to move between levels so work can get done

Bell-LaPadula

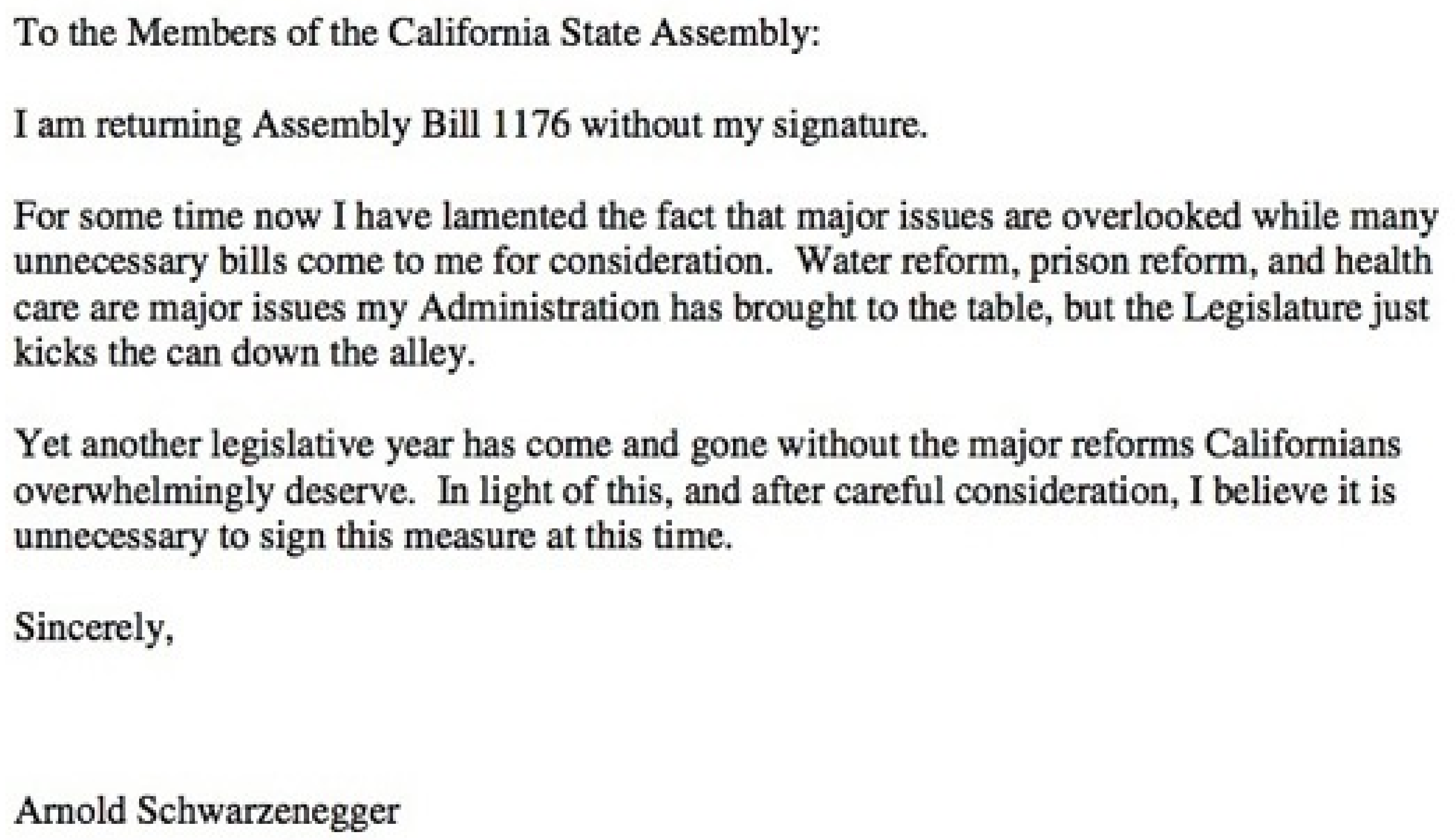
- Designed to formally model the US Department of Defense security policy
- Users and data are assigned security levels from low to high
- Then we have the policy:
  - Users at a lower level cannot read information at a higher level (**no read up**)
  - Users at a higher level cannot send information to a lower level (**no write down**)

Covert Channels

- Do these models actually guarantee that in practice, unauthorized information can't leak between levels?
- **No!** It turns out there are many more ways to transfer information than the authorized channels
- For example, a user at one security level could send sensitive information to a lower level by performing more or less computation
  - Resulting delays could be observable to lower levels, transmitting information

Steganography

- Even if we allow a human to inspect all data passed between levels, we can still have covert channels
- The way to do this is with *steganography* – hiding data inside other data
- Essentially, make changes that are not observable unless you know what you're looking for, and use those changes to encode information



To the Members of the California State Assembly:

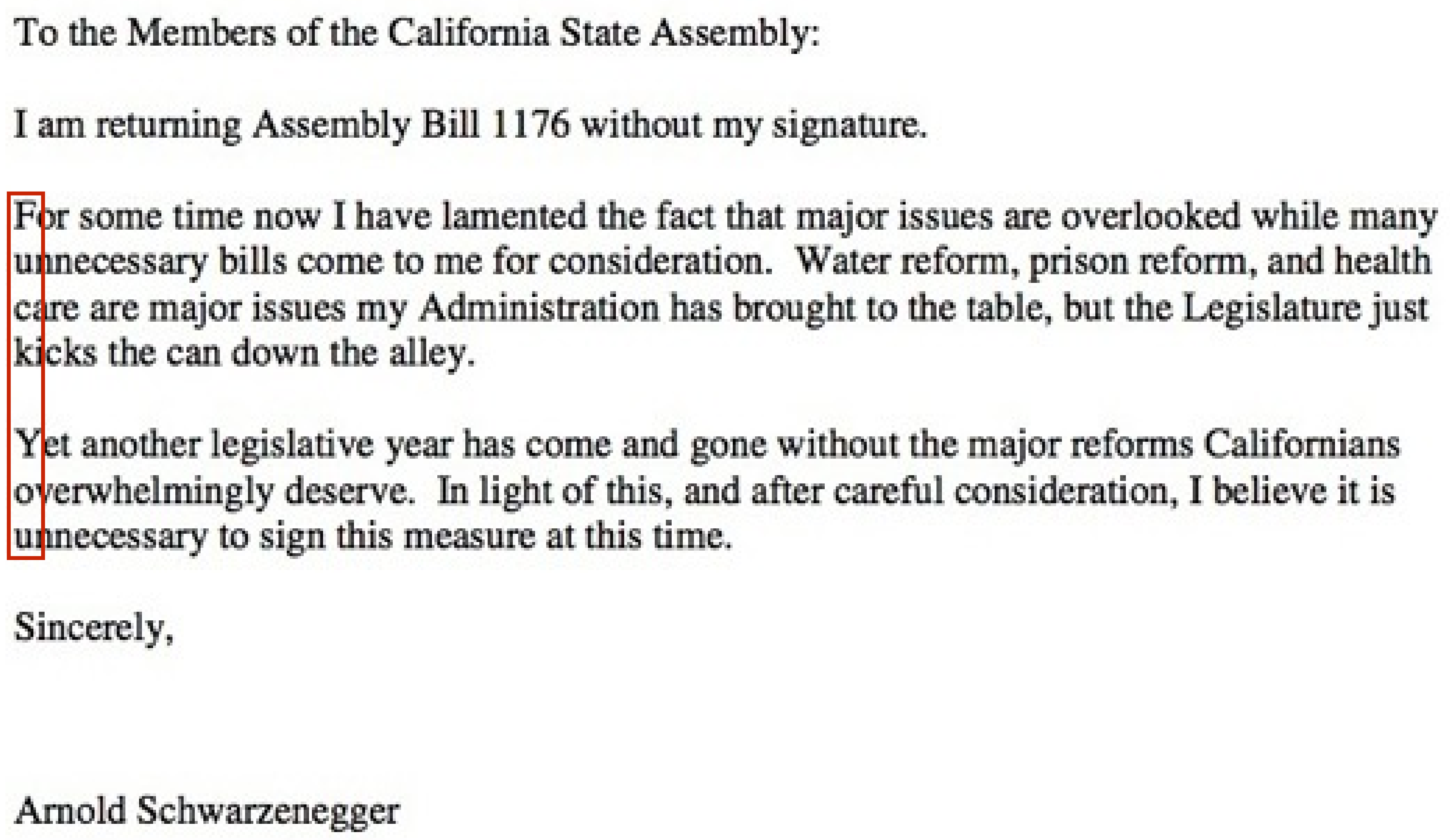
I am returning Assembly Bill 1176 without my signature.

For some time now I have lamented the fact that major issues are overlooked while many unnecessary bills come to me for consideration. Water reform, prison reform, and health care are major issues my Administration has brought to the table, but the Legislature just kicks the can down the alley.

Yet another legislative year has come and gone without the major reforms Californians overwhelmingly deserve. In light of this, and after careful consideration, I believe it is unnecessary to sign this measure at this time.

Sincerely,

Arnold Schwarzenegger



To the Members of the California State Assembly:

I am returning Assembly Bill 1176 without my signature.

For some time now I have lamented the fact that major issues are overlooked while many
unnecessary bills come to me for consideration. Water reform, prison reform, and health
care are major issues my Administration has brought to the table, but the Legislature just
kicks the can down the alley.

Yet another legislative year has come and gone without the major reforms Californians
overwhelmingly deserve. In light of this, and after careful consideration, I believe it is
unnecessary to sign this measure at this time.

Sincerely,

Arnold Schwarzenegger

Steganography

- How does this work? Suppose each color is represented by three bytes (red, green, blue)

- Take the least significant bit of each byte and use it to store your own information

- This only adds or subtracts 1 from each color value – not visually detectable

- So we can now get 1 bit of secret information per byte of public information!

Security

- The Security Environment
- OS Security
- Controlling Access to Resources
- Formal Models of Secure Systems
- ➡ Basics of Cryptography

Cryptography

- Cryptography has been around a *long* time – at least since the Greeks, and probably earlier
- But it has changed a lot since then
- Particularly since World War II, many advances have been made in both codemaking and codebreaking
- (Right now, the codemakers are winning!)

Kerckhoffs's Principle

- Basic principle: it should not matter if the cryptographic *algorithm* is known by everyone
- The only secret information should be a *secret key* chosen by the participants
  - Less information to keep secret
  - If the key is revealed, we can just change it
- Restated by Claude Shannon: "The enemy knows the system"

Secret Key Cryptography

- Two parties who want to communicate agree on a *secret key* shared between them
- Then they use *encryption* and *decryption* functions:
  - E(Data, Key) = Encrypted Data
  - D(Data, Key) = Decrypted Data
  - D(E(Data, Key), Key) = Data

Example: Caesar Cipher

- Algorithm: take each letter and shift it forward in the alphabet by n letters

- Secret key: the number to shift by (1-25)

- For example, ROT13: shift by 13 letters

  - ATTACK AT DAWN

  - NGGNPX NG QNJA

Secret Key Cryptography

- Modern secret key crypto is much more sophisticated
- Algorithms like AES (the Advanced Encryption Standard) employ substitutions and permutations so that the output has no relationship to input unless the secret is known
  - As far as we know! We have been wrong before...
- Keys are generally 128 or 256 **random** bits – much too large to try all combinations

Public Key Cryptography

- Secret key cryptography was the only kind that existed until the 1970s
- It's inconvenient! Two parties have to somehow securely transmit (or agree on) a secret key
- Particularly on the Internet, this is impractical
    - Imagine having to visit an Amazon office to get a secret key in order to shop online...

Public Key Cryptography

- Instead of a single secret key, we now have a key with two parts: a public key and a private key
  - E(PubKey, Data) = Encrypted Data
  - D(PrivKey, Data) = Decrypted Data or Data
  - **D(PrivKey, E(PubKey, Data)) = Data**
- So now to communicate securely, we just need to know someone's private key

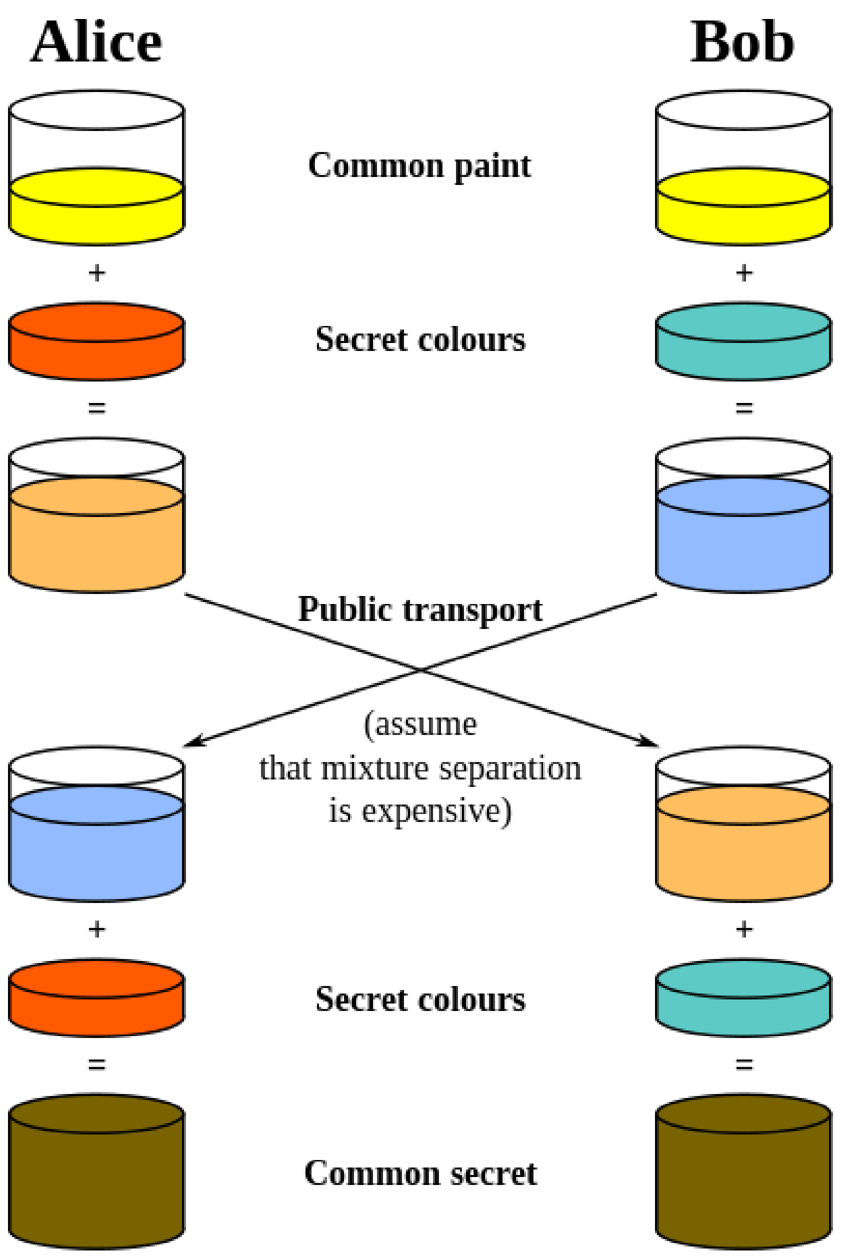
Public Key Crypto Algorithms

- Some examples:
  - RSA (1977)
  - Elliptic Curve Algorithms

Key Exchange

- A slightly different twist on this: can we use only public communications to allow two people to agree on a secret key?
- Surprisingly, the answer is yes!
- Whitfield Diffie and Martin Hellman first showed this was possible in 1976

Diffie-Hellman by Analogy



A.J. Han Vinck, Introduction to public key cryptography, p. 16

Hash Functions

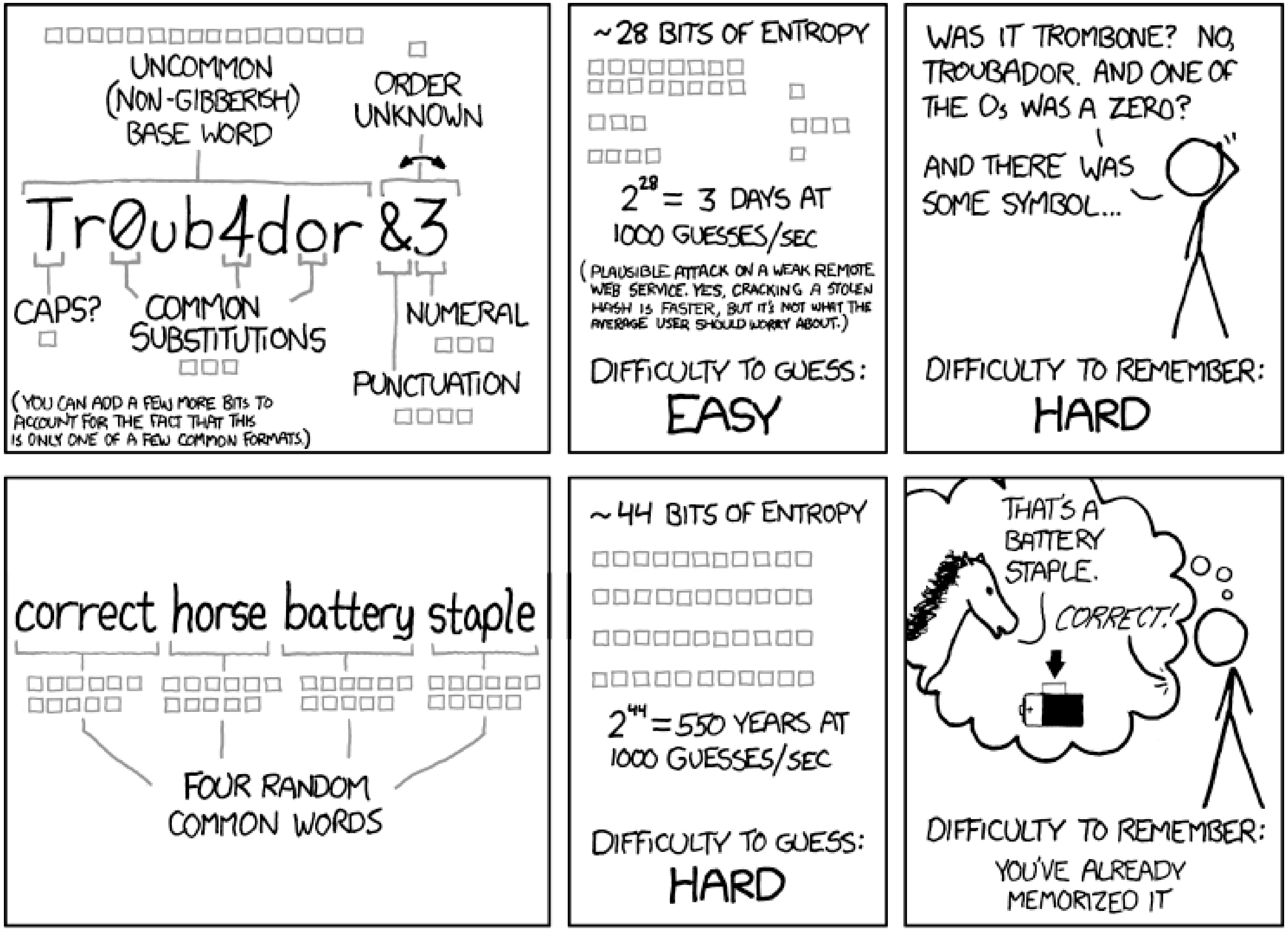
- We saw before that one-way functions can let us do useful things like store capabilities
- Basic idea: Hash(x) = y
  - **Preimage resistance:** It should be very difficult to take y and figure out x
  - **Second preimage resistance:** It should be very difficult to find another value z where Hash(z) = y
  - **Collision resistance:** It should be very difficult to find x1 and x2 such that Hash(x1) = Hash(x2)
- Examples: MD5, SHA-1, SHA-3

Authentication

- To prove you are who you say you are, we usually use one of three things:
    - Something you know
    - Something you have
    - Something you are

Passwords

- "Something you know" – one of the simplest and oldest forms of authentication, and still ubiquitous
- Generally easy to change
- Very easy to implement, since you can just ask for a short string and compare it with the one you have stored


UNCOMMON (NON-GIBBERISH) BASE WORD
ORDER UNKNOWN
Tr0ub4dor&3
CAPS?
COMMON SUBSTITUTIONS
NUMERAL
PUNCTUATION
(YOU CAN ADD A FEW MORE BITS TO ACCOUNT FOR THE FACT THAT THIS IS ONLY ONE OF A FEW COMMON FORMATS.)
~28 BITS OF ENTROPY
2^28 = 3 DAYS AT 1000 GUESSES/SEC
(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)
DIFFICULTY TO GUESS: EASY
WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE Os WAS A ZERO?
AND THERE WAS SOME SYMBOL...
DIFFICULTY TO REMEMBER: HARD
correct horse battery staple
FOUR RANDOM COMMON WORDS
~44 BITS OF ENTROPY
2^44 = 550 YEARS AT 1000 GUESSES/SEC
DIFFICULTY TO GUESS: HARD
THAT'S A BATTERY STAPLE.
CORRECT!
DIFFICULTY TO REMEMBER: YOU'VE ALREADY MEMORIZED IT

THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

Password Storage

- It's generally a bad idea to store passwords directly
- If someone gets ahold of the password file, they can read all passwords
- This is particularly bad since people tend to re-use their passwords in multiple places...

Password Storage

- Instead, we use a cryptographic hash function on the password
- We compute (for example):
  - SHA1("goodpassword") = 7e5ce399fbe3713ec7f6aae370448cdf990f0aaa
- Then we only store 7e5ce399fbe3713ec7f6aae370448cdf990f0aaa
- Now to check the password someone entered, we hash it and compare the hashes instead

How Passwords are Broken

- Assuming they only have a file with password hashes, how can an attacker find out the original passwords?

- Password hashes nowadays are broken primarily in two ways:

    - **Precomputation**: Spend a lot of time beforehand computing the hash of every possible password, then just look up each hash in your table

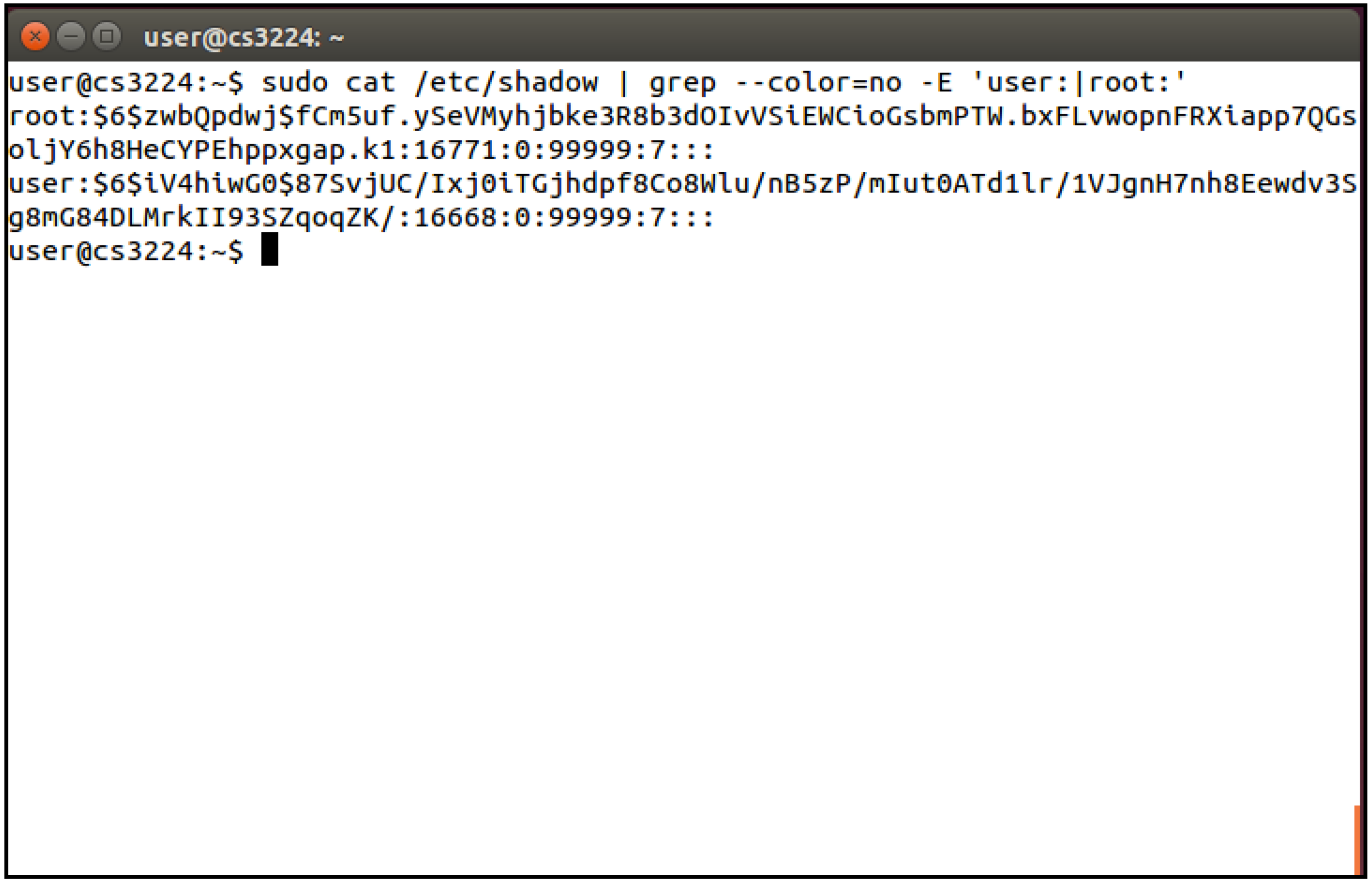
    - **Dictionary Attacks**: Guess the most common passwords (e.g. using a dictionary), hash them, and compare

Password Salts

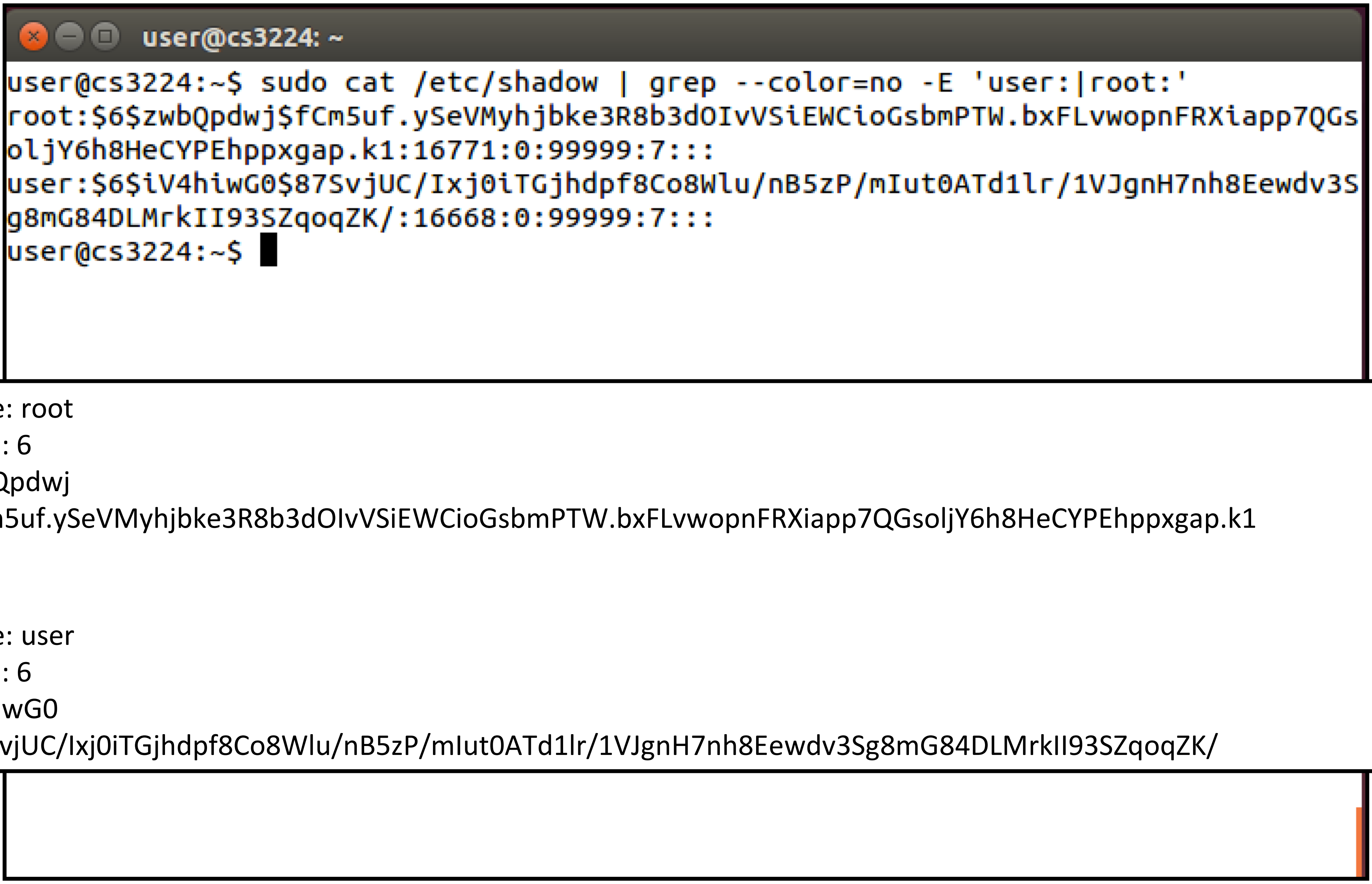
- To avoid precomputation attacks, we can use a *salt*
- Instead of computing and storing Hash(Password), we instead:
    - Generate a random string (a **salt**)
    - Compute Hash(Salt+Password)
    - Store on disk Salt, Hash(Salt+Password)
- Now to do a precomputation, you have to store not just the hash of every possible password (hard but doable) but the hash of every possible password+salt (way too much space)

UNIX Password Files

- UNIX password files are hashed and stored with salts:



```
user@cs3224:~$ sudo cat /etc/shadow | grep --color=no -E 'user:|root:'
root:$6$zwbQpdwj$fCm5uf.ySeVMyhjbke3R8b3dOIvVSiEWCioGsbmPTW.bxFLvwopnFRXiapp7QGs
oljY6h8HeCYPEhppxgap.k1:16771:0:99999:7:::
user:$6$iV4hiwG0$87SvjUC/Ixj0iTGjhdpf8Co8Wlu/nB5zP/mIut0ATd1lr/1VJgnH7nh8Eewdv3S
g8mG84DLMrkII93SZqoqZK/:16668:0:99999:7:::
user@cs3224:~$
```



```
user@cs3224:~$ sudo cat /etc/shadow | grep --color=no -E 'user:|root:'
root:$6$zwbQpdwj$fCm5uf.ySeVMyhjbke3R8b3dOIvVSiEWCioGsbmPTW.bxFLvwopnFRXiapp7QGs
oljY6h8HeCYPEhppxgap.k1:16771:0:99999:7:::
user:$6$iV4hiwG0$87SvjUC/Ixj0iTGjhdpf8Co8Wlu/nB5zP/mIut0ATd1lr/1VJgnH7nh8Eewdv3S
g8mG84DLMrkII93SZqoqZK/:16668:0:99999:7:::
user@cs3224:~$
```

Username: root
Algorithm: 6
Salt: zwbQpdwj
Hash: fCm5uf.ySeVMyhjbke3R8b3dOIvVSiEWCioGsbmPTW.bxFLvwopnFRXiapp7QGsoljY6h8HeCYPEhppxgap.k1

Username: user
Algorithm: 6
Salt: iV4hiwG0
Hash: 87SvjUC/Ixj0iTGjhdpf8Co8Wlu/nB5zP/mIut0ATd1lr/1VJgnH7nh8Eewdv3Sg8mG84DLMrkII93SZqoqZK/

Time- and Memory-Hard Password Hashing

- Cryptographic hash functions were developed to be *fast*
- This is good for many uses of them, but bad for passwords: it lets an attacker try **billions** of passwords per second
- To avoid this, people have developed hash algorithms that are deliberately very slow (and cannot be sped up)
  - One example is **bcrypt**
- We may also want it to take a lot of memory
  - A newer algorithm called **scrypt** does this