Security for Operating Systems: Cryptography, Authentication, and Protecting OS Resources

CS 111

Operating Systems

Peter Reiher

Outline

- Basic concepts in computer security
- Design principles for security
- Important security tools for operating systems
- Access control
- Cryptography and operating systems
- Authentication and operating systems
- Protecting operating system resources

Cryptography

- Much of computer security is about keeping secrets
- One method of doing so is to make it hard for others to read the secrets
- While (usually) making it simple for authorized parties to read them
- That's what cryptography is all about

What Is Encryption?

- Encryption is the process of hiding information in plain sight
- Transform the secret data into something else
- Even if the attacker can see the transformed data, he can't understand the underlying secret
- Usually, someone you want to understand it can

Cryptography Terminology

- Typically described in terms of sending a message
  - Though it's used for many other purposes
- The sender is *S*
- The receiver is *R*
- *Encryption* is the process of making message unreadable/unalterable by anyone but *R*
- *Decryption* is the process of making the encrypted message readable by *R*
- A system performing these transformations is a *cryptosystem*
  - Rules for transformation sometimes called a *cipher*

Plaintext and Ciphertext

- *Plaintext* is the original form of the message (often referred to as *P*)

Transfer $100 to my savings account

- *Ciphertext* is the encrypted form of the message (often referred to as *C*)

Sqzmredq #099 sn lx rzuhmfr zbbntms

Cryptographic Keys

- Most cryptographic algorithms use a *key* to perform encryption and decryption
  - Referred to as *K*
- The key is a secret
- Without the key, decryption is hard
- With the key, decryption is easy
- Reduces the secrecy problem from your (long) message to the (short) key
  - But there's still a secret

More Terminology

- The encryption algorithm is referred to as *E()*
- $C = E(K,P)$
- The decryption algorithm is referred to as *D()*
- The decryption algorithm also has a key
- The combination of the two algorithms are often called a *cryptosystem*

Symmetric and Asymmetric Cryptosystems

- Symmetric cryptosystems use the same keys for E and D :

  $P = D(K, C)$

  – Expanding, $P = D(K, E(K,P))$
- Asymmetric cryptosystems use different keys for E and D:

  $C = E(K_E,P)$

  $P = D(K_D,C)$

  – Expanding, $P = D(K_D , E(K_E ,P))$

Desirable Characteristics of Keyed Cryptosystems

- If you change only the key, a given plaintext encrypts to a different ciphertext
- Same applies to decryption
- Changes in the key ideally should cause unpredictable changes in the ciphertext
- Decryption should be hard without knowing the key
- The less a given key is used, the better (in security terms)

Cryptography and Operating Systems

- What does cryptography have to offer operating systems?
- Which hard security problems in operating systems can we solve with cryptography?
- Where doesn't it help?

Cryptography and Secrecy

- Pretty obvious for networks
- Only those knowing the proper keys can decrypt an encrypted message
  - Thus preserving secrecy
- Used cleverly, it can provide other forms of secrecy
- Clear where we'd use this for distributed systems
- Where does it make sense in a single machine?

Cryptography and Authentication

- How can I prove to you that I created a piece of data?
- What if I give you the data in encrypted form?
  - Using a key only you and I know
- Then only you or I could have created it
  - Unless one of us told someone else the key . . .
  - Or one of us is trying to screw the other

Cryptography and Integrity

- Changing one bit of a piece of ciphertext completely garbles it
  - For many forms of cryptography
- If a checksum is part of encrypted data, that's detectable
- If you don't need secrecy, can get the same effect
  - By encrypting only the checksum

Symmetric Cryptosystems

- $C = E(K,P)$
- $P = D(K,C)$
- $E()$ and $D()$ are not necessarily the same operations

Advantages of Symmetric Cryptosystems

+ Encryption and authentication performed in a single operation
+ Well-known (and trusted) ones perform much faster than asymmetric key systems
+ No centralized authority required
  - Though key servers help a lot

Disadvantages of Symmetric Cryptosystems

- Encryption and authentication performed in a single operation
  - Makes signature more difficult
- Non-repudiation hard without servers
- Key distribution can be a problem
- Scaling
  - Especially for Internet use

Some Popular Symmetric Ciphers

- The Data Encryption Standard (DES)
  - The old US encryption standard
  - Still fairly widely used, due to legacy
  - Weak by modern standards
- The Advanced Encryption Standard (AES)
  - The current US encryption standard
  - Probably the most widely used cipher
- Blowfish
- There are many, many others

Symmetric Ciphers and Brute Force Attacks

- If your symmetric cipher has no flaws, how can attackers crack it?
- *Brute force* – try every possible key until one works
- The cost of brute force attacks depends on key length
  - Assuming random choice of key
  - For N possible keys, attack must try N/2 keys, on average, before finding the right one

How Long Are the Keys?

- DES used 56 bit keys
  - Brute force attacks on that require a lot of time and resources
  - But they are demonstrably possible
  - Attackers can thus crack DES, if they really care
- AES uses either 128 bit or 256 bit keys
  - Even the shorter key length is beyond the powers of brute force today
  - 2^{127} decryption attempts is still a lot, by any standard

Asymmetric Cryptosystems

- Often called *public key cryptography*
  - Or PK, for short
- The encrypter and decrypter have different keys
  - $C = E(K_E, P)$
  - $P = D(K_D, C)$
- Often works the other way, too
  - $C' = E(K_D, P)$
  - $P = D(K_E, C')$

Using Public Key Cryptography

- Keys are created in pairs
- One key is kept secret by the owner
- The other is made public to the world
  - Hence the name
- If you want to send an encrypted message to someone, encrypt with his public key
  - Only he has private key to decrypt

Authentication With Public Keys

- If I want to "sign" a message, encrypt it with my private key
- Only I know private key, so no one else could create that message
- Everyone knows my public key, so everyone can check my claim directly
- Much better than with symmetric crypto
  - The receiver could not have created the message
  - Only the sender could have

PK Key Management

- To communicate via shared key cryptography, key must be distributed
  - In trusted fashion
- To communicate via public key cryptography, need to find out each other's public key
  - "Simply publish public keys"
- Not really that simple, for most cases

Issues With PK Key Distribution

- Security of public key cryptography depends on using the right public key
- If I am fooled into using wrong one, that key's owner reads my message
- Need high assurance that a given key belongs to a particular person
  - Either a *key distribution infrastructure*
  - Or use of *certificates*
- Both are problematic, at high scale and in the real world

The Nature of PK Algorithms

- Usually based on some problem in mathematics
  - Like factoring extremely large numbers
- Security less dependent on brute force
- More on the complexity of the underlying problem

Choosing Keys for Asymmetric Ciphers

- For symmetric ciphers, the key can be any random number of the right size
  - You can't do that for asymmetric ciphers
- Only some public/private key pairs "work"
  - Generally, finding a usable pair takes a fair amount of time
  - E.g., for RSA you perform operations on 100-200 digit prime numbers to get keys
- You thus tend to use one public/private key pair for a long time
  - Issues of PK key distribution and typical usage also suggest long lifetimes for these keys

Example Public Key Ciphers

- RSA
  - The most popular public key algorithm
  - Used on pretty much everyone's computer, nowadays
- Elliptic curve cryptography
  - An alternative to RSA
  - Tends to have better performance
  - Not as widely used or studied

Security of PK Systems

- Based on solving the underlying problem
  - E.g., for RSA, factoring large numbers
- In 2009, a 768 bit RSA key was successfully factored
- Research on integer factorization suggests keys up to 2048 bits may be insecure
  - In 2013, Google went from 1024 to 2048 bit keys
- Size will keep increasing
- The longer the key, the more expensive the encryption and decryption

Combined Use of Symmetric and Asymmetric Cryptography

- Very common to use both in a single session
- Asymmetric cryptography essentially used to "bootstrap" symmetric crypto
- Use RSA (or another PK algorithm) to authenticate and establish a *session key*
- Use DES or AES with session key for the rest of the transmission

For Example,

Alice wants to share K_S only with Bob

Bob wants to be sure it's Alice's key

Only Bob can decrypt it

Only Alice could have created it

Alice

K_{EA} K_{DA}

K_{DB}

Bob

K_{EB} K_{DB}

K_{DA}

K_S

$C=E(K_S,K_{DB})$

$M=E(C,K_{EA})$

$K_S=D(C,K_{EB})$

$C=D(M,K_{DA})$

Authentication for Operating Systems

- What is authentication?
- How does the problem apply to operating systems?
- Techniques for authentication in operating systems

What Is Authentication?

- Determining the identity of some entity
  - Process
  - Machine
  - Human user
- Requires notion of identity
  - One implication is we need some defined name space
- And some degree of proof of identity

Where Do We Use Authentication in the OS?

- Typically users authenticate themselves to the system
- Their identity tends to be tied to the processes they create
  - OS can keep track of this easily
- Once authenticated, users (and their processes) typically need not authenticate again
  - One authentication per session, usually
- Distributed systems greatly complicate things

Authentication Mechanisms

- Something you know
  - E.g., passwords
- Something you have
  - E.g., smart cards or tokens
- Something you are
  - Biometrics
- Somewhere you are
  - Usually identifying a role

Passwords

- Authentication by what you know
- One of the oldest and most commonly used security mechanisms
- Authenticate the user by requiring him to produce a secret
  - Usually known only to him and to the authenticator

Problems With Passwords

- They have to be unguessable
  - Yet easy for people to remember
- If sent over the network, susceptible to password sniffers
- Unless fairly long, brute force attacks often work on them

Handling Passwords

- The OS must be able to check passwords when users log in
- So must the OS store passwords?
- Not really
  - It can store an encrypted version
- Encrypt the offered password
  - Using a *one-way function*
  - E.g., a secure hash algorithm like SHA1
- And compare it to the stored version

Is Encrypting the Password File Enough?

- What if an attacker gets a copy of your password file?
- No problem, the passwords are encrypted
  - Right?
- Yes, but . . .

Dictionary Attacks

Harpo	2st6'sG0
Zeppo	G>I5{as3
Chico	
Karl	sY(34,ee
Groucho	
Gummo	3(;wbnP]

sY(34,ee

abaca is Karl Marx's password!

Rats!!!!

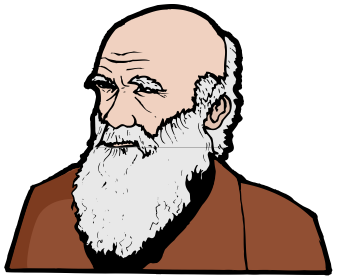
Now you can hack the Communist Manifesto!

Salted Passwords

- A technique to combat dictionary attacks
- Combine the plaintext password with a random number
  - Then run it through the one-way function
- The random number need not be secret
- It just has to be different for different users
- You store the salt integer with the password
  - Generally in plaintext

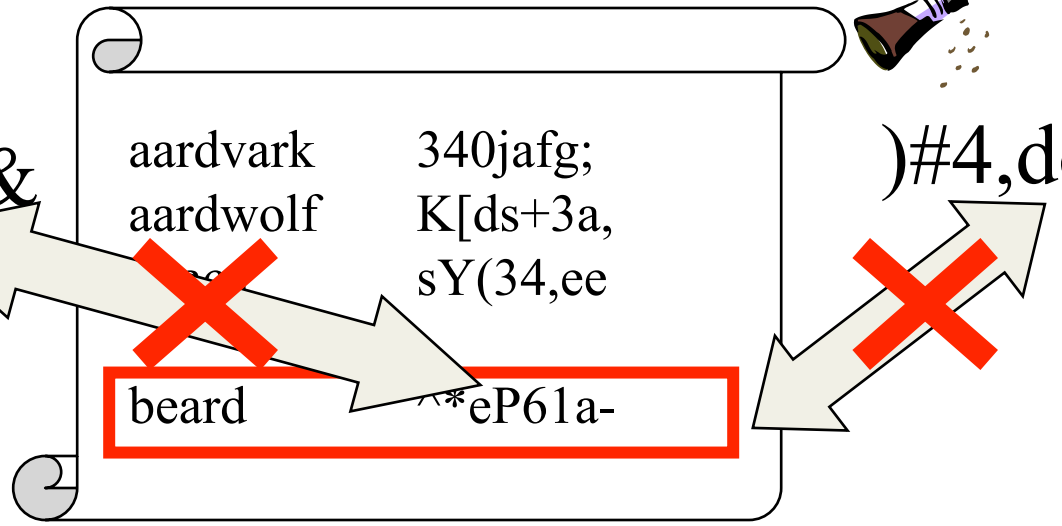
Did It Fix Our Problem?

Karl Marx



Charles Darwin

D0Cls6&



)#4,doa8

Are My Passwords Safe Now?

- If I salt and encrypt them, am I OK?
- Depends on the quality of the passwords chosen
- Attacker can still perform dictionary attacks on an individual password, with its salt
- If the password isn't in the dictionary, no problem
- If it is, the attack succeeds
- Which is why password choice is important

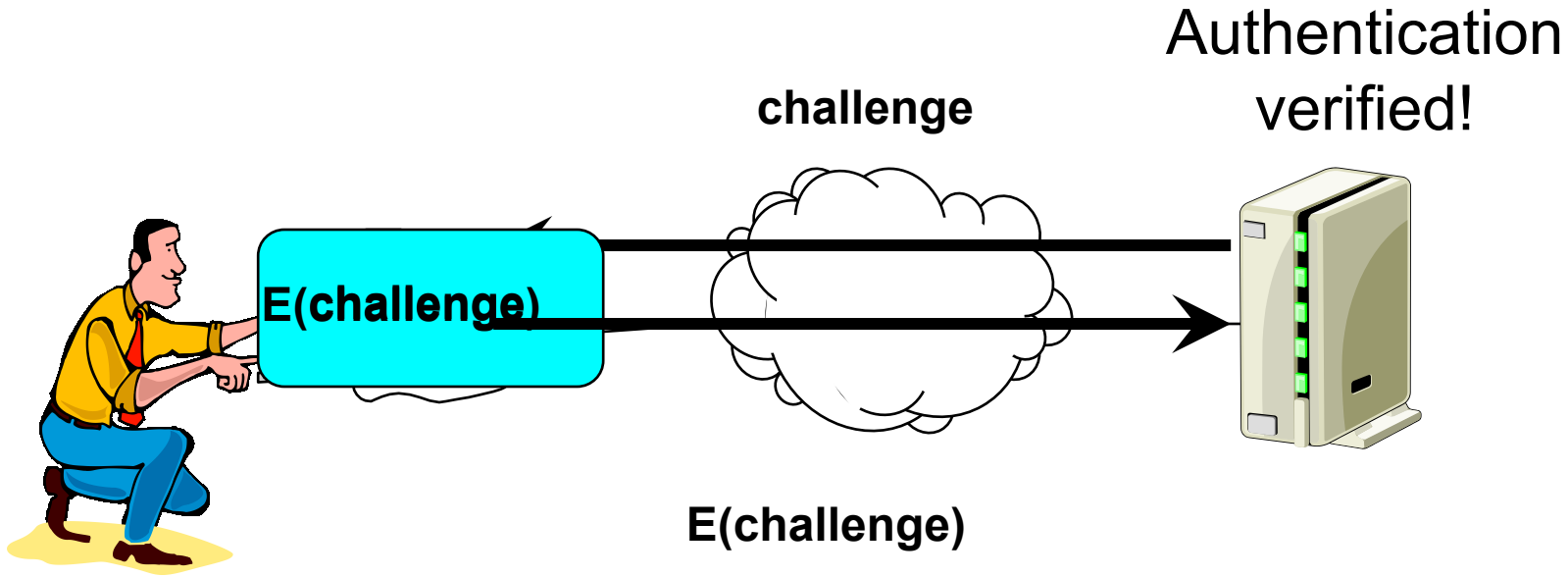
Password Selection

- Generally, long passwords chosen from large character sets are good
- Short passwords chosen from small character sets are bad
- How long?
  - A matter of time
  - Moore's law forces us to make them ever longer
- What's a large character set?
  - Upper and lower case letters, plus numbers, plus symbols (like ^ and @)

Authentication Devices

- Authentication by what you have
- A smart card or other hardware device that is readable by the computer
  - Safest if device has some computing capability
  - Rather than just data storage
- Authenticate by providing the device to the computer
- More challenging when done remotely, of course

Authentication With Smart Cards



How can the server be sure of the remote user's identity?
By proper use of cryptography

Problems With Authentication Devices

- If lost or stolen, you can't authenticate yourself
  - And maybe someone else can
  - Often combined with passwords to avoid this problem
- Unless cleverly done, susceptible to sniffing attacks
- Requires special hardware
- There have been successful attacks on some smart cards

Biometric Authentication

- Authentication based on who you are
- Things like fingerprints, voice patterns, retinal patterns, etc.
- To authenticate, allow the system to measure the appropriate physical characteristics
- Biometric measurement converted to binary and compared to stored values
  - With some level of match required

Problems With Biometrics

- Requires very special hardware
- May not be as foolproof as you think
- Many physical characteristics vary too much for practical use
  - Day to day or over long periods of time
- Generally not helpful for authenticating programs or roles
- What happens when it's cracked?
  - You only have two retinas, after all

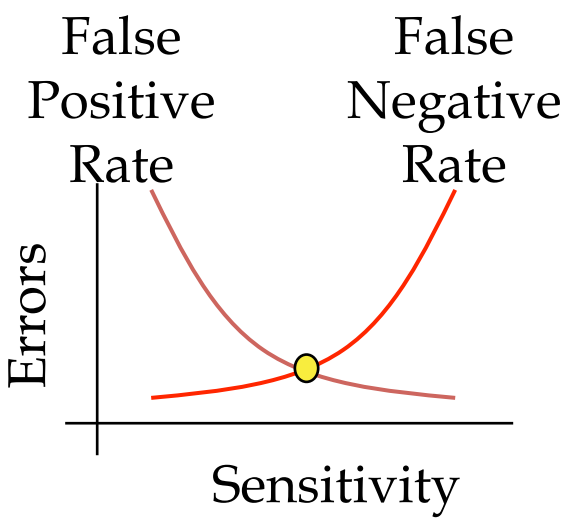
Characterizing Biometric Accuracy

How many false positives?

Match made when it shouldn't have been

Versus how many false negatives?

Match not made when it should have been



The Crossover Error Rate (CER)

Generally, the higher the CER is, the better the system

Retinal scans: 1:10,000,000+

Fingerprint readers: 1:500

Protecting Operating Systems Resources

- How do we use these various tools to protect actual OS resources?
- Memory?
- Files?
- Devices?
- IPC?
- Secure booting

Protecting Memory

- Most modern operating systems provide strong memory protection
- Usually hardware-based
- Most commonly through use of page tables and paging hardware
- Each process can only access page frames mapped in its own page table
- Reduces issue to OS' proper use of page tables for processes

Protecting Files

- We've already discussed this
- Most file systems have a built-in access control model
- The OS must enforce it
- All file access done through system calls
- Which gives the OS a chance to enforce the access control policy
- Typically checked on open
  - Issue of complete mediation . . .

A File Data Vulnerability

- What if someone bypasses the operating system?
- Directly accessing the disk as a device
- The OS typically won't allow that to happen
  - If it's still in control . . .
- But there can be flaws or misconfigurations
- Or the disk can be moved to another machine
  - Which may not enforce the access permissions it specifies

Full Disk Encryption

- FDE
- A solution to this problem
- Encrypt everything you put on the disk
- Decrypt data moved from the disk to memory
- Can be done in hardware
  - Typically in the disk drive or controller
- Or software
  - Typically by the operating system
- Various options for storing the key

Protecting Devices

- Most devices are treated as files
- So the file protection model applies
- In some cases, some parts of the devices are memory mapped into processes
  - Memory protections apply, here
  - But potential issues if you map them into more than one process
- Non-OS controlled bus interfaces can also cause problems (e.g., Firewire)

Protecting IPC

- IPC channels are often also treated like files
- So the same protection model and mechanisms apply
- Even shared memory is handled this way
  - But especially important to remember that you don't get complete mediation here
  - And granularity of protection is the segment, not the word or page or block

Secure Boot

- Our OS-based protection mechanisms rely on one fundamental assumption
  - We are running an OS that properly implements them
- What if we aren't running the OS that we think we are?
- Then all bets are off
- The false OS can do whatever it wants
- So we need to be sure we've booted what we wanted to boot

The Bootstrap Process

- When a computer is powered on, the OS is not usually resident in memory
- It gets put there by a *bootstrap loader*
- The bootstrap program is usually very short
- Located in an easily defined place
- Hardware finds it, loads it, runs it
- Bootstrap then takes care of initializing the OS

Booting and Security

- Most systems make it hard to change bootstrap loader
  - But it must have enough flexibility to load different OSes
  - From different places on machine
- Malware likes to corrupt the bootstrap
- Trusted computing platforms can help secure bootstrapping

Approaches to Bootstrap Security

- TPM – an industry standard
- A hardware-assisted method to guarantee that the right bootstrap was loaded
  - And, from that, guarantee that the right OS was booted
  - And possibly build up further security from that
- SecureBoot – a Microsoft technology
- Built into the boot hardware and SW
- Essentially, only allows booting of particular OS versions

Conclusion

- This scratches the surface of security issues for the OS
- Ideally, OS design and implementation should consider security from start to finish
- Ongoing research looks at improving OS security
  - E.g., by proving security properties of the kernel
- Since the OS is the foundation of the other software, its security is crucial