CS 458: Computer Security and Privacy

Michael Noukhovitch

Spring 2016, University of Waterloo

Notes written from Erinn Atawater's lectures.

Contents

1 Introduction

1.1 Security

Security can be defined as:

confidentiality access to systems is limited to authorized

integrity getting the correct data

availability system is there when you want it

1.2 Privacy

There are many definitions but we will stick to **informational self-determination**, where you control the information about you

1.3 Terminology

assets things we want to protect

vulnerabilities weaknesses in a system that can be exploited

threats loss or harm that may befall a system

- interception
- interruption
- modification
- fabrication

threat model set of threats to defend against (who/what)

attack an action which exploits a vulnerability to execute a threat

control removing or reducing a vulnerability

1.4 Types of Defence

Defend against an attack:

- **prevent** stop the attack from happening
- **deter** make the attack more difficult
- **deflect** make it less attractive for attacker
- **recover** mitigate effects of the attack

Make sure that defence is correct with principles:

- **easiest penetration** system is only as strong as weakest link
- **adequate protection** don't spend more on defence than the value of the system

1.5 Methods of Defence

- Software controls: passwords, virus scanner . . .

- Hardware controls: fingerprint reader, smart token . . .

- Physical controls: locks, guards, backups . . .

- Policies: teaching employees, password changing rules

2 Program Security

2.1 Flaws, faults, and failures

2.1.1 Defintions

flaw problem with a program

fault a potential error inside the logic

failure an actual error visible by the user

2.1.2 Unexpected Behaviour

A spec will list the things a program will do but an implementation may have additional behaviour. This can cause issues as these behaviours might not be tested and would be hard to test.

2.2 Unintentional Security Flaws

2.2.1 Types of Flaws

- **intentional**

  - **malicious**: inserted to attack system
  - **nonmalicious**: intentional features meant to be in the system but can cause issues

- most flaws are **unintentional**

2.2.2 Buffer Overflow

Most common exploited type of security flaw when program reads or writes past the bounds of the memory that it should use. If the attacker exploits it they can override things like the *saved return address*. Targets programs on a local machine that run with setuid priveleges or network daemons

Example 2.1. basic buffer overflow

```
#define LINELEN 1024

char buffer[LINELEN];
strcpy(buffer, argv[1]);
```

Types:

- only a single byte can be written past the end of the buffer

- overflow of buffers on the heap (instead of the stack)

- jump to other parts of the program or libraries (instead of shellcode)

Defences:

- language with bounds-checking

- non-executable stack (mem is never both writable and executable)

- stack at random virtual addresses for each process

- "canaries" detect if stack has been overwritten before return

2.2.3 Integer Overflows

Program may assume integer is always positive, and below certain value. Overflow will make a too large singed integer negative, violating assumptions.

2.2.4 Format String Vulnerabilities

Format strings can have unexpected consequences, `printf`

- `buffer` parse buffer for `\%s` and use whatever is on the stack to process found format params

- `%s%s%s%s` may crash your program

- `%x%x%x%x` dumps parts of the stack

- `%n` will write to an address on the stack

2.2.5 Incomplete Mediation

mediation ensure what the user has entered is a meaningful request

incomplete mediation application accepts incorrect user data

Though **client-side** mediation is helpful to the user, you should always perform **server-side** mediation

- check values entered by user

- check state stored by client

TOCTTOU "time of check to time of user" errors are race conditions that may affect correct access to resources

Defend by making all access control information **constant** between TOC and TOU

- keep private copy of request

- act on object itself as opposed to symlinks . . .

- use locks on object

2.3 Malware

- written with malicious intent

- needs to executed to cause harm

2.3.1 Virus

virus malware that infects other files (with copies of itself)

infect modify existing program (*host*) so opening gives control to virus

payload end goal of virus (e.g. corrupt, erase . . .)

Protection can come in two forms:

signature-based keep a list of all known viruses (but how to deal with polymorphic viruses?)

behaviour-based look for suspicious system behaviour

2.3.2 Worm

worm self-containing piece of code that replicated with little/no user input

- often use security flaws in widely deployed software

- searches for other unprotected sources to spread to

2.3.3 Other Types

trojan horse claim to do something normal, but hide malware

scareware scaring user into agreeing

ransomware ransoming user's resource

logic bomb written by insider, already on your computer waiting to be triggered

2.4 Other Malicious Code

2.4.1 General Attacks

web bug tiny object in web page, fetched from a different server that can track you

backdoor instructions set to bypass normal authentication, come from

- forgetting to remove

- testing purposes

- law enforcement

- malicious purposes

salami attack attack from many smaller attacks

privelege escalation raises privelege of attacker, can cause legitimate higher privelege code to execute attack (e.g. `-froot` attack)

rootkit tool to gain priveleged access and then hide itself

- cleans logs

- modify basic commands `ls`. . .

- modify kernel so no programs can see it

2.4.2   Man-in-the-middle

intercepts communication but passes it on to intended party eventually

keystroke logger logs keyboard input and spies on user

- application-specific

- system logger (all keystrokes)

- hardware logger (physical device)

interface illusions tricks user to execute malicious action with UI

> **phishing** make fake website look real to extract user information

2.5   Nonmalicious flaws

covert channels transfer data through secret/non-standard channel (e.g. hide data in published report)

side channels attack based on knowledge from physical behaviour of computer

- RF emissions

- power consumption

- cpu usage

- reflection of the screen

2.6   Security Controls

2.6.1   Design

Design programs so they're less likely to have flaws

- modularity

- encapsulation

- information hiding

- mutual suspicion

- confinement/sandboxing

2.6.2 Implementation

When actually coding, reduce security flaws

- don't use C
- static code analysis
- formal methods
- genetic diversity (run varied code)
- educate yourself

2.6.3 Change Management

Make sure that all changes to the code maintain security

- track changes in a system (CVS ...)
- do post-mortems of security flaws
- code reviews
    - guided code reviews
    - easter-egg code reviews (intentional flaws)

2.6.4 Testing

Make sure implementation meets specification *and nothing else*

black box testing treat code as an opaque interface

fuzz testing submit completely random data

white box testing testing which understands how it works, good for regression testing

2.6.5 Documentation

For posterity, write down:

- choices made
- things that didn't work
- security checklist

2.6.6 Maintenance

Make sure that code out there gets better not worse

standards rules to incorporate controls at each software stage

process formal specs of how to implement each standard

audits externally verify your processes are correct and followed

3 Operating System Security

3.1 Protection in a General-Purpose System

3.1.1 Overview

Protect a user from attacks, and protect resources:

- CPU
- memory
- I/O devices
- programs
- data
- networks
- OS

3.1.2 Separation/Sharing

Keep one users' objects separate from others'

- **physical** use different physical resource
- **temporal** execute at different times
- **logical** give impression that no other users exist
- **cryptographic** enrypt data to make it unintelligible

OS' can allow for **flexible sharing**, not "all or nothing"

3.1.3 Memory Protection

Prevent one program from corrupting other programs, OS, data...

fence register exception if memory access below address in fence register

base/bounds register pair exception if memory not between register pair

tagged architechture each memory word has extra bits that identify access to that word

3.1.4 Segmentation

segmentation each program has different mem segments for code, data, stack. Virtual address contains `<segment name, offset within segment>` and segment name is mapped to physical address in *Segment Table*

- + each address reference is checked for protection
- + different levels of protection
- + share/restrict access to a segment
- - external fragmentation
- - costly: dynamic length out-of-bounds checks, segment names

3.1.5 Paging

pages equal divisions of virtual address space

frames equal divisions of physical memory (size same as page)

page table maps page number to corresponding frame `<page #, offset within page>`, also contains memory protection bits for each page

+ each address reference is checked for protection

+ share/restrict access to a page with different rights

+ unpopular pages can be moved to disk

- internal fragmentation

- infeasible to have different levels of protection for different data

3.1.6 x86

Includes both segmentation and paging, with memory protection bits:

- no access
- read access
- read/write access
- *no execute

3.2 Access Control

3.2.1 Overview

We need to protect more than just memory, with three goals

- check every access
- enforce least privelege
- verify acceptable use

Create an **access control matrix** for a set of protected objects 0, subjects S, and rights R (r,w,x,o).

3.2.2 Access Control Methods

Access Control Lists each object has a list of subjects and their access rights

Capability unforgeable token that gives own some access rights to an object

RBAC admin assign users to roles and grants access rights to roles

- can be hierarchical
- a user can have multiple roles for different tasks
- **separation of duty** same person can't be responsible for two different roles on a task

3.3   User Authentication

3.3.1   Overview

Computers systems have to identify and authenticate users before authorizing them

identify who are you

authenticate prove your identify

3.3.2   Authentication Factors

Four classes, something the user:

- **knows** password, PIN

- **has** ATM card, physical key

- **is** biometrics

- **context** location, time

Using multiple factors ("two-factor" authentication) improves security if they are different types

3.3.3   Passwords

Attacks:

- shoulder surfing

- keystroke logging

- phishing

- password re-use

- password guessing

User defenses:

- choosing good passwords (long and not easily guessable)

- hygiene

    - write down > store insecurely
    - change regularly
    - site specific passwords
    - don't reveal
    - don't enter sensitive info on public computers

Admin defenses:

- store only cryptographic hashes

salt user-specific addition to hash (based on time of day . . .)

pepper salt not stored alongside password

- use expensive hash

  – SHA-x take microseconds
  – bcrypt takes hundreds of milliseconds
  – scrypt also uses a lot of memory

- use a **MAC** message authentication code

  – also uses secret key to compute fingerprint
  – impossible to crack without secret key
  – as secure as hashing if key does leak

- make password recovery force a reset

  – don't email them the password, since you shouldn't be storing it

- one-time passwords

  – fight interception attacks
  – **challenge-response** to generate the password

3.3.4 Biometrics

Authenticate user if *physical characteristic* is sufficiently similar to stored trait, but this has issues

- remote authentication can't test if you are trying to bypass it

- since we check for similarity, not equality, false positives are more likely

- facial recognition software still not good enough for security

- privacy issue if your stored biometrics leak

- if leaked, you can change a password, but not biometrics

- some of your biometrics are not secret (easy to get photos of. . .)

3.4 Security Policies and Models

3.4.1 Trusted OS

We trust an OS if we have confidence that it provides security services which build on four factors:

policy set of rules outlining what is secured and why

model implementes the policy and can be used for reasoning about it

design spec of how OS implements model

trust assurance that OS is implemented according to design

3.4.2 Trusted Software

Software that does what we expect it to do and nothing more

functional correctness software works correctly

enforcement of integrity wrong inputs don't impact correctness

limited privelege access rights are minimized

appropriate confidence level rated as required

3.4.3 Security Policies

Basic military model:

- each object has a sensitivity level

- each object is assigned to one or more compartments

- subject can access if it *dominates* the requirements

Chinese Wall security policy that once you access info you can't access info about competitors

ss-property read access by subject to object if each object previously accessed is either from the same company, or a different type of company (no conflict)

*-property write access by subject to object if all readable object by subject are either from the same company or have been *sanitized*

lattice security model where there is a unique upper and lower bound for any two points, i.e. each level is distinct

3.4.4 Security Models

Bell-La Padula regulates information flow in lattices, so users get information only with their clearance

- no read up (read more secure documents)

- no write down (write less secure documents)

Biba prevent inappropriate modification of data

- subjects and objects ordered by integrity

- no read down (don't contaminate reliable person with unreliable info)

- no write up (don't contaminate reliable info with unreliable person)

Low Watermark Property instead of enforcing integrity rules, just reduce integrity when it is violated

- **subject** if subject s reads object o, then $I(s) =$ greatest lower bound of $I(s)$ and $I(o)$

- **object** if a subject writes to an object, the reduce integrity of object to greatest lower bound

3.5 Trusted OS Design

3.5.1 Security Design Principles

least privelege use least priveleges possible

economy of mechanism protection mechanism should be simple

open design avoid security by obscurity (secret keys not algorithms)

complete mediation check everything

permission based default is no permissions

separation of priveleges two or more conditions to get access

ease of use make it easy or no one will use it

3.5.2 Security Features

identification/authentication access control

- mandatory: central authority determines access
- directionary: owners of objects have some control over who can access
- role-based: central authority defines roles

object reuse protection

- stored data should be inaccessible to next user
- deleting a file should actually wipe it (not hidden!)

complete mediation

- all accesses must be checked

trusted path

- defend against illusions
- assure that keystrokes and mouse movements are sent correctly

accountability and audit

- log all security-related events
- find good middle-ground for granularity of logs

intrusion detection

- correlated actual behaviour with "normal" behaviour
- alarm if behaviour looks abnormal

3.5.3   Trusted Computing Base

TCB part of the trusted OS that is necessary to enforce OS policies

security kernel runs between OS and hardware maintaining security

rings security level where processes can only access rings that level or above

reference monitor monitor with collection of access controls that must be tamperproof, unbypassable, and analyzable

virtualization way to provide logical separation/isolation

- memory: page mapping to give separate address space

- machines: virtualize I/O, files, printers . . .

3.5.4   Least Privelege

chroot sandbox/jail a command by changing its root directory

compartmentalization split application into parts and apply least privelege to each part

setuid bit causes executable to run under identity of owner not caller

- careful of **confused deputy** attack where you convince program another user is executing a setuid'd program

3.5.5   Assurance

Convince others to trust out OS through

- testing

- formal verification

- validation (requirements . . .)

Can also have third party evaluate it based on

- **Orange Book** security ratings from DoD

- **common criteria** international effort, protection profiles against security threats and objectives

4   Network Security

4.1   Network Concepts

4.1.1   Overview

Internet has

- decentralized control

- uncontrollable traffic flow through different nodes

- different types of nodes/links

TCP/IP suite covers (abridged)

- link: ethernet, wifi . . .

- network: IP . . .

- transport: TCP, UDP . . .

- application: HTTP, FTP, SSL . . .

Protocols were based on *trust* and *non-malicious* intent

4.2 Threats

4.2.1 Port Scan

Each application runs on a **port** and unsecure systems (**loose lips**) can reveal what is running on each port, allowing the attacker to target weakest application

- nmap tool can identify application

- login application can reveal OS, version . . .

4.2.2 Intelligence

Find information out in the open:

- social engineering

- dumpster diving

- eavesdropping

- google searching

- facebook searching

4.2.3 Wiretapping

Owner of a node can mointor all communication through it

- copper cable

  - inductance allows for eavesdropping without physical contact
  - cable cutting splicing otherwise

- fiber optic

  - no inductance, and splicing is possible but maybe detectable
  - cable "bending"

- satellite

– signal path is wide so nearby attacker can eavesdrop

- wifi

  – easily intercepted nearby or far away using directed antenna

  – need authentication for basic security

- LAN

  – attacker can un-ignore wrongly delivered packets using a **packer sniffer**

4.2.4 Impersonation

Impersonate by stealing a password

- guessing

- exploit default password

- sniff for password between nodes

- social engineering

and exploit trust beween machines and accounts where `rhosts/rlogin` can allow another use on another machine to act as themselves on their machine without having to enter a password

4.2.5 Spoofing

spoofing when one object masquerades as another

- **url spoofing** uwaterlo.ca

- **evil twin** attacks wifi access points using the same name

- **session hijacking** stealing cookies or session key to hijack

- **man-in-the-middle** attacker becomes stealth intermediate node

4.2.6 Traffic Analysis

Discovering the *existance* of communication between two parties (e.g. whistleblower) can be sensitive, and can be discovered by attacker looking at send and recieve addresses in headers

4.2.7 Integrity Attacks

Package modification (as they are being transmitted)

- change payload

- change sender/reciever

- replay previously seen packets ("buy 10 stocks" over and over)

- delete or create packets

- TCP will can use a checksum but that can be easily defeated

DNS cache poisoning

- feed wrong mappings to system redirecting them to your address

- DNNSEC created to combat this

4.2.8 Protocol Failures

TCP and other protocols can be attacked directly:

- assumes all nodes implement protocol faithfully

- optional requests of systems to help flow control (can be ignored)

- may not check if packet is well formatted

- may include broken security mechanisms (e.g. WEP)

4.2.9 Website Vulnerabilities

- website defacement

- send malicious URL to web server (to exploit)

- submit modified state (cookie, session id) to exploit incomplete mediation

- code injection

    XSS steal sensitive information on page and send it to attacker
    CSRF perform malicious action on some website (e.g. bank) if user is logged in there

4.2.10 Denial of Service

- cutting a wire, jamming a signal

- **ping flood** overload ability to respond to pings

- **smurf attack** spoof sender of ping as victim, and have return pings all go there

- **syn flood** overflow memory with stored SYN packets

- send packet fragments that can't be assembled

- send packets that are all hashed into same bucket

- **black hole attack** router advertises very low cost for destination and then drops all packets

- **DNS cache poisoning** make packets route to wrong host

4.2.11 Distributed Denial of Service

DDoS makes it difficult to stop the source of the attack by using lots of machines

zombie/bot computer controlled with malware

botnet network of computers used to DDoS

amplification use nodes that respond with more data than they are queried

reflection send queries with victim's address as source

SNMP simple network management protocol on many home routers which is usually vulnerable

Botnets can be controlled in many ways:

- central node: all bots connect and it gives commands

- p2p distributing updates

- fast flux: single host name maps to hundreds of addresses, constantly swap in/out of DNS to make tracking difficult

- domain generation algorithm: generate a large set of domain and contact a tue subset of them for instructions

4.2.12 Active Code

Server can ask client to execute code on its behalf

- Java

- Javascript

- ActiveX

- Flash

Can be dangerous depending on sandboxing, "trusted" source

4.3 Network Security Controls

4.3.1 Segmentation and Separation

- have servers on different machines

- segment by functional and access requirements

- split by vulnerability (inside/outside firewall)

4.3.2 Redundancy

- avoid single points of failure

- different software, genetic diversity

- sync redundant servers closely (so they can easily take over)

4.3.3 Access Controls

- router ACL: drop packets with particular source and address
    - expensive for high traffic
    - difficult to gather logs
    - source addresses can be spoofed
- firewalls: filter based on other critera than addresses

4.4 Firewalls

4.4.1 Overview

- all traffic goes through these choke points, **firewalls**
- carefully examine traffic, possibly refuse it access
- does not protect against attacks from inside

4.4.2 Types

packet filtering gateways make decision based on header of a packet, can drop spoofed traffic

stateful inspection keep state to identify packets that belong together

application proxies app-specific firewall with full knowledge about communication and sophisticated processing

forward proxy accessing server outside the company

reverse proxy accessing company from outside

personal home computer with a whitelist, protect against attacks on servers

Demilitarized Zone subnet containing external services with internal and exteranl firewall

4.5 Honeypots

4.5.1 Overview

- set up unprotected computer as a trap for attacker
- computer should have no activity
- monitor for any activity as sign of break-in
- continue monitoring honeypot to learn about new threats

4.5.2   Types

low interaction daemon that emulates running host(s)

- \+ easy to install/maintain

- \- limited amount of information gathered

- \- easier to detect

high interaction deploy real hardware/software, tracking with stealth network switches
and keyloggers

- \+ can capture lots of information

- \+ can caputure unexpected behaviour

- \- complex

4.6   Intrusion Detection Systems

4.6.1   Overview

IDS monitor activity to identify malicious events

- recieve events from sensors

- store and analyze

- take action

4.6.2   Host-based

Run on a host to protect this host

- \+ can exploit lots of information

- \- miss out on information of other hosts

- \- if host gets subverted, so does IDS (single point of failure)

4.6.3   Network-based

Run on a dedicated node to protect all attached hosts

- \+ difficult to subvert

- \- has to rely on information available in packets

Distributed IDS combination of host-based and network-based

4.6.4   Signature-based

Detect attack "signatures"

- \+ may exploits statistical analysis to catch tough stuff

- \- may fail if attacker modifies attack (polymorphic worms)

4.6.5 Heuristic/Anomaly-based

Look for behaviour that is out of the ordinary

- model good behaviour and alert if system doesn't follow model

- model bad behaviour and alert if system resembles model

- classify as

  - good
  - suspicious
  - unknown

- learn to classify unknown events over file

- high rate of false positives

- can be fooled by sophisticated attack

Two step process:

1. train on examples of normal behaviour (while definitely not infected)

2. test on running behaviour of vulnerable machine

5 Internet Application Security and Privacy

5.1 Cryptography

5.1.1 Overview

cryptanalysis breaking secret messages

cryptography making secret messages

- confidentiality: prevent eavesdropping

- integrity: prevent message modification

- authenticity: prevent impersonation

5.1.2 Kerckhoffs Principle

The security of a cryptosystem should not rely on a secret that's hard (or expensive) to change

- "The enemy knows the system" - Claude Shannon

- system only as secure as number of keys

Don't use a secret encryption *method* but instead have a large class of public encryption methods and use a secret key to choose one

5.1.3   Attacker's Information

The attacker (eavesdropping) may know

- algorithm (public class of encryption method)

- some part of the plaintext

- corresponding plaintext/cipher pairs

- access to encryption/decryption oracle

5.2   Secret-key Encryption

5.2.1   Overview

symmetric encryption same key used to encrypt is used to decrypt

one-time pad key is random bitstring same length as plaintext, XOR'd, used only once

Need a secure medium to share the secret key between two parties

5.2.2   Computational Security

Most security is "imperfect" and can be cracked with enough effort: trying every key

- 40-bit crypto: old US legal export limit, 18 hours on 1 computer

- 56-bit crypto: DES, 134 years

- 128-bit crypto: modern standard, $635 * 10^{21}$ years

5.2.3   Ciphers

stream cipher pseudorandom key used to XOR plaintext

- RC4 is used today

- can be very fast

- tricky to use correctly (same key for two different messages. . .)

block cipher operate on message one block at a time (64 or 128 bits)

- AES is modern implementation

- **mode of operation** what to do with multiple blocks
    - ECB: encrypt each next block separately, you'll be able to see pattern
    - CBC, CTR, GCM: use initial value as salt, don't expose pattern

5.3 Public-key Encryption

5.3.1 Overview

asymmetric cryptography different keys for encryption, decryption

- publicize your public key
- other people send you messages encrypted with your public key
- use private key to decrypt their messages

Issues:

- shortcuts to crack (unlike secret-key) e.g. 2^{33} work to crack 128-bit RSA
- takes a long time to encrypt

5.3.2 Hybrid cryptography

hybrid cryptography combine speed of secret key with asymmetry of public key

- pick random 128-bit key for secret-key encryption
- encrypt message with secret key
- encrypt secret key with public key of reciever
- send reciever secret-keyed message and public-keyed secret key

5.4 Integrity

5.4.1 Overview

Traditionally use **checksum** to verify message hasn't changed, but Mallory can usually easily change the message to maintain the checksum, so we need a **cryptographic checksum** (hard to find different message with same checksum)

The method of integrity checking is just like encyrption and depends on key/secret:

- no key: cryptographic hash
- secret key: MAC
- public key: signing

5.4.2 Cryptographic Hash

Without requiring a key we can use a **cryptographic hash** to compute $y = h(x)$, called **message digest**

- preimage-resistance: given y, hard to find x
- second preimage-resistance: given x, hard to find x' where $h(x') = h(x)$
- collision resistance: hard to find distinct x_1, x_2 where $h(x_1) = h(x_2)$

Hash the message and send the message digest, checking the recieved message with the hash, but it only work is there is a secure way of sending the message digest

5.5 Authentication

5.5.1 Message Authentication Codes

For secret key encryption, use **MAC** which :

- large class of hash functions, **MAC**s

- use shared secret key to pick correct one

- only if you know secret key, can you generate or check computed hash (**tag**)

- SHA-1-HMAC, CBC-MAC

Combine with a cipher, usually Encrypt-then-MAC

5.5.2 Digital Signing

If we want to *publicly* prove who sent it, **non-repudiatiation**, then use digital signatures

- sign a message with a private **signature key**

- verify the message with their public **verification key**

Signing vs public key cryptography

- hybrid signatures also available: signing a hash of the message

- sign your public keys to make them trustworthy

- signing keys are long-lived, encryption keys are short-lived

5.5.3 Certificate Authorities

Key management is difficult, how to securely find a verification key:

- **manual keyring** know it personally, SSH

- **web of trust** trust a friend, PGP

- **certificate authority** trust a third party, TLS

Certificate Authority a third party that keeps a directory of verification keys

- a user sends the CA their verification key and signed personal information

- CA ensures information is correct

- CA generates and *signs* a **certificate** of user's personal information and verification key

Everyone should have a copy of a CA verification key and each CA can issue certificates for CAs in levels below them (**public key infrastructure**)

5.6 Security Controls

5.6.1 Program and OS Security

Secret key cryptography can be difficult to defend, so use public key if local machine only needs access to public part: decryption and signature verification

- only install app if it was signed by vendor (BB)

- only upgrade app if it was signed by developer (Android)

- only allow signed executables (iOS)

5.6.2 Encryption

Encrypted code

- processor only runs encrypted code

- decryption is processor-dependent

Encrypted data

- disk drive encryption to defend when stolen/lost

- doesn't protect against other users

5.7 Link-layer Security

5.7.1 Overview

Intended to protect LAN

- WEP

- WPA

- WPA2

5.7.2 WEP

Wired Equivalent Protocol terrible protocol, should do

- confidentiality (prevent from reading your traffic)

- access control (prevent from using your wifi)

- data integrity

but doesn't do either of those things

- sender and reciever share secret k (40 or 104 bits)

- computer checksum $c(M)$

- pick random IV v and generate keystream $RC4(v, k)$

- ciphertext = keystream $\oplus < M, c(M) >$

- transmit v and ciphertext

Many issues:

- v is only 24 bits long, and the only part of the encryption that changes

- checksum used is linear and easily defeated

- plaintext/ciphertext pair is given away by response, and allows for easy injection

- plaintext/ciphertext enough to get key and decrypt messages

5.7.3 WPA/WPA2

Wifi Protected Access replaced WEP as a short-term patch

- replaced CRC-32 with real MAC

- IV is 48 bits

- key changed frequently

WPA2 802.11i standard of WPA

- replace RC4 and MAC with CCM authenticated encryption mode (with AES)

- strong except in PSK (pre-shared key) mode

5.8 Network-layer Security

5.8.1 VPN

Virtual Private Network security across networks (end-to-end)

- send through local VPN gateway

- gateway encrypts and sends to remote gateway (e.g. tunnelling)

- remote gateway decrypts and sends along

Tunnelling sending the message of one protocol in payload of another, out of sequence

- IP-over-TCP, because usually it is TCP being sent over IP

- PPP-over-DNS, link layer over application layer

5.8.2   VPN Style

IPsec standard VPN setup

- **transport** mode

    - connecting a single laptop to a home network
    - only content of original IP packet is encrypted

- **tunnel** mode

    - connecting two networks
    - contents *and header* of original IP packet encrypted and authenticated
    - result placed inside new IP packet going to remote gateway

Microsoft PPTP older protocol with design flaw like WEP

Tunnel PPP IP-over-PPP-over-ssh-over-TCP-over-IP

OpenSSH IP-over-SSH tunnelling directly

5.9   Transport-layer

5.9.1   Security

TLS standardized SSL for protecting HTTP connections

- client tells server its ciphersuites

- server responds with certificate

    - host name
    - verification key
    - admin info
    - signature from CA

- server chooses ciphersuite

- client validates certificate

- client and server run agreement protocol

    - establish key for symmetric encryption
    - choose MAC algorithm from ciphersuite

- now they communicate using symmetric encryption and MAC

TLS provides

- server auth

- message integrity

- message confidentiality

- client auth (optional)

5.9.2 Privacy

Tor the onion router, allows making TCP connection without revealing IP

- connect to three onion routers sequentially, creating keys $k1, k2, k3$
- send a message M encrypted as $E_{k1}(E_{k2}(E_{k3}(M)))$, which gets decrypted along the way
- each node only knows the previous and next nodes, not sender or reciever
    - the website only communicates with node 3
    - node 3 communicates with node 2 and the website
    - node 2 communicates with node 1 and node3
    - node 1 communicates with you and node 2
- replies are encrypted sequentially so that you recieve $E_{k1}(E_{k2}(E_{k3}(M)))$

Tor provides anonymity that is

- unlinkable in the long term (connecting tomorrow is likely using different nodes)
- linkable in the short term (refresh will use the same nodes)

5.9.3 Nymity

Continuum of **nymiyty**

- verinymity: ID, SIN card . . .
- persistent pseudonymity: username on a website
- linkable anonymity: prepaid phone card
- unlinkable anonymity: cash transaction, Tor

Easier to get to a higher level from lower, so always start as low as possible

5.10 Application-layer Security and Privacy

5.10.1 Remote Login

SSH secure shell

- client connects
- server sends verification key
- client and server run agreement protocol to establish session keys
- client authenticates to server
- server accepts auth, login proceeds

Two ways to authenticate

- send password over encrypted channel (server knows hash of password)
- sign random challenge with private signature key (server needs to know your public signature key)

5.10.2 Email

PGP pretty good privacy, public-key email encryption

- hybrid encyrption of email messages
- digital signatures on email messages

Obtaining others' public keys and signatures keys is difficult

- find **fingerprint**, cryptographic hash of the key
- double check fingerprints with the user or someone that knows them
- having a web of users verifying each other is called **web of trust**

5.10.3 Instant Messaging

Perfect forward secrecy future key compromises should not reveal past communication

- use a session key created using Diffie-Hellman
- discard the session key after use
- use long-term keys to authenticate Diffie-Hellman

Deniable authentication authenticate the other person, but can't convince a third party who sent the message

- shared-key authentication to compute the MAC
- use MAC, so auth is deniable

Implementations:

- **Off-The-Record Messaging** IM providing confidentiality, deniability
- **Signal Protocol** extends deniability using **DAKE** (triple Diffie-Hellman key exchange)

6 Database Security and Privacy

6.1 Introduction

relational db structured queryable collection of records

record data consisting of elements

elements fields of a record

schema structure set by database admin

subschema result of a query

DBMS database management system, provides support for queries/management

relational model store records in one or multiple tables (**relations**)

- each table has columns (**attributes**) and rows (**tuples**)
- tables can have relations between each other

6.2 Security Requirements

6.2.1 Database Integrity

logical protect against db corruption

physical recover from physical problems

- perform backups
- keep transaction logs

6.2.2 Element Integrity

element integrity ensure correctness/accuracy of elements

access control limit who can update

element checks validate correctness

- prevents mistakes by users
- enforced by triggers
- e.g. must be numeric, no more than one president

changelog/shadow fields keep track to undo accidental changes

error detection codes protect against OS or HDD problems

updates must be atomic and two-phase (gather info, make change) to ensure integrity

6.2.3 Referential Integrity

Each table has

primary key minimal set of attributes to unique identify each tuple (userid)

foreign key primary keys of other tables

referential integrity no dangling foreign keys (for each foreign key, there is a corresponding entry as a primary key)

6.2.4 Auditability

- keep an audit log of all db accesses
- retroactively identify who accessed forbidden data
- decide on granularity of logging

6.2.5 Access Control

- control at relation, record, or element level

- control all ops (select, insert . . .)

- maybe consider past queries (can give more info than intended)

- maybe consider type of query

6.2.6 User Authentication/Availability

- database may do own authentication

- facilitate sharing but make sure availability doesn't suffer from concurrent access

6.3 Data Disclosure and Inference

6.3.1 Data Disclosure

Types:

- exact data

- bounds

- negative result (e.g. num of felonies != 0)

- existence

- probable value

Balance needs of:

security forbid any queries that access sensitive data, even if aggregate non-sensitive

precision aggregated result should reveal as much non-sensitive data as possible

6.3.2 Inference Attack

inference deriving sensitive data from (supposedly) non-sensitive data

direct attack directly yield sensitive data (e.g. obfuscate query)

indirect attack infer sensitive data from statistical result, tracker attack

Controls:

suppression suppress sensitive data from result

- n-item k-percent rule: if a subset of n records is responsible for over k percent of the result, omit the n records

concealing answer is close to actual value but not exact

- combined result: report set or range of possible values

- random sample: compute result on random sample of database

- random data perturbation: add/subtract small random error before computing result

extra:

- query analysis: maintain history of user's queries and observe possible inferences

6.3.3 Tracker Attack

If DBMS refuses to answer queries that match $< k$ or $> N - k$ records, we can use a **tracker** to find a query C using unrelated query T and query everything S

$$\text{If } C \text{ matches} < k: \qquad q(C) = q(C \vee T) + q(C \vee \neg T) - q(S)$$
$$\text{If } C \text{ matches} > N - k: \qquad q(C) = 2 * q(S) - q(\neg C \vee T) - q(\neg C \vee \neg T)$$

6.3.4 Differential Privacy

- query reponse should not depend on individual being/not being part of the dataset

- query k has η-differential privacy if for all datasets D and D' differing by one row, probability $K(D)$ has a specific output is at most $e^{\eta} *$ probabilty $K(D')$ has this output $(0 \leq \eta \leq 1)$

- achieve this by adding noise to result

6.3.5 Data Aggregation

Aggregating less sensitive results can form sensitive results, and it can be done by multiple people together outside the DBMS, tough to stop.

6.4 Multilevel Security Databases

6.4.1 Definition

- classification of information by confidentiality (confidential/not confidential)

- granularity of classification (can be down to element-level)

- each object is classified

6.4.2 *-Property

Implementing *-property is difficult

- write-up without reading is tough

- write-downs need sanitation

- trusted processes can do anything

6.4.3   Approaches

confidentiality different answers for query depending on clearance

- hiding existance can mean having to allow duplicate records at different clearances (*polyinstatiation*)

partitioning separate DBs for each classification level

+ simple

- data redundancy in multiple DBs

- high level user needs combined high + low level data

encryption encrypt data with a key unique to classification level

• use securely: same value in different records should have different ciphertext

- processing is expensive because of decryption

integrity lock provide integrity and access control, use data items:

• actual data

• integrity level (concealable)

• crypto signature (or MAC) for above and item name and record number

+ any db can be used to store data item + integrity locks

+ trusted procedure handles access control + manages locks, e.g. enforce *-property

- takes up more space

- makes processing more expensive

- must encrypt (on top of this!) if there are other ways to access db

6.5   Design of Secure Databases

A trusted front-end:

• authenticates user and forwards query to DBMS

• gets result from DBMS and remove items user shouldn't see

• allows use of exisiting DBMS and databases

commutative filters rewrite user query according to user's clearance (remove attributes, add constraints)

• discard forbidden data items early on

• may still need to do post-processing

distributed/federated db forwards user query only to DBs user can access

- may need to combine results

- doesn't scale to many different clearance levels

views logival database that is a subset of another database

- elements can correspond to an element or combination (e.g. sum)

- show user only rows they're allowed to see

- set `UNDEFINED` any elements they're not allowed to see

truman semantics pretend that everything user sees is everything that exists, don't fail queries just return incorrect results

6.6   Data Mining and Data Release

6.6.1   Security Problems

confidentiality

- can reveal sensitive info by aggregating different sources,

- companies can sell data to each other

integrity

- mistakes in data can lead to negative impact on people (e.g. no fly list mixup)

- difficult to correct mistakes if data has been widely shared

- false negatives also dangerous

availability

- naming conventions make combining databases difficult (e.g. last name, name, family name)

- must distinguish between inability to find data and inability to find correlation

privacy

- data mining may reveal sensitive information

- use **quasi-identifiers** that don't allow for reidentification

- laws (PIPEDA, HIPAA) should control collection, use, disclosure

- secret gov't orgs are tougher to control

6.6.2 Privacy Preservation Metrics

k-**anonymity** for each record there are at least $k-1$ other records from which this one cannot be distinguished

- **homogeneity attack** if the k rows are identical (e.g. both for the k-anonymous and for actual field, e.g. disease) then knowing that someone is in the table means you know what their field is (e.g. Bob is in table, Bob is (902**,196*-*-*-*), Bob has cancer)

- **background knowledge attack** knowing background info allows you to guess which of possible sensitive fields it is (even if there are > 1)

l-**diversity** for any quasi-identifier, there should be l "well-represented" values of the sensitive field

t-**closeness** ensure the distributions of values for any quasi-identifier are within t of the distribution for the whole table

6.6.3 Privacy Preserving Methods

value swapping swapping values of some data fields for a subset of released records

- linking to other data can no longer be completely accurate

- makes strong correlations less strong, makes weak correlations go away

adding noise adding small postive or negative error to each value

- given distribution of noisy data and distribution of noise, can calculate distribution of data (but not actual data)

- protects privacy without sacrificing accuracy

sampling release only a subset of data with geographic coarsening and top/bottom coding

- **geographic coarsening** restrict geographic identifiers to those containing a certain pop. (e.g 100k people)

- **top/bottom coding** replace top/bottom identifier with ranges (e.g. $90, 95 \rightarrow \geq 90$)

synthetic data build a distribution model based on gathered data and generate synthetic data with similar characteristics