Web and Database Security

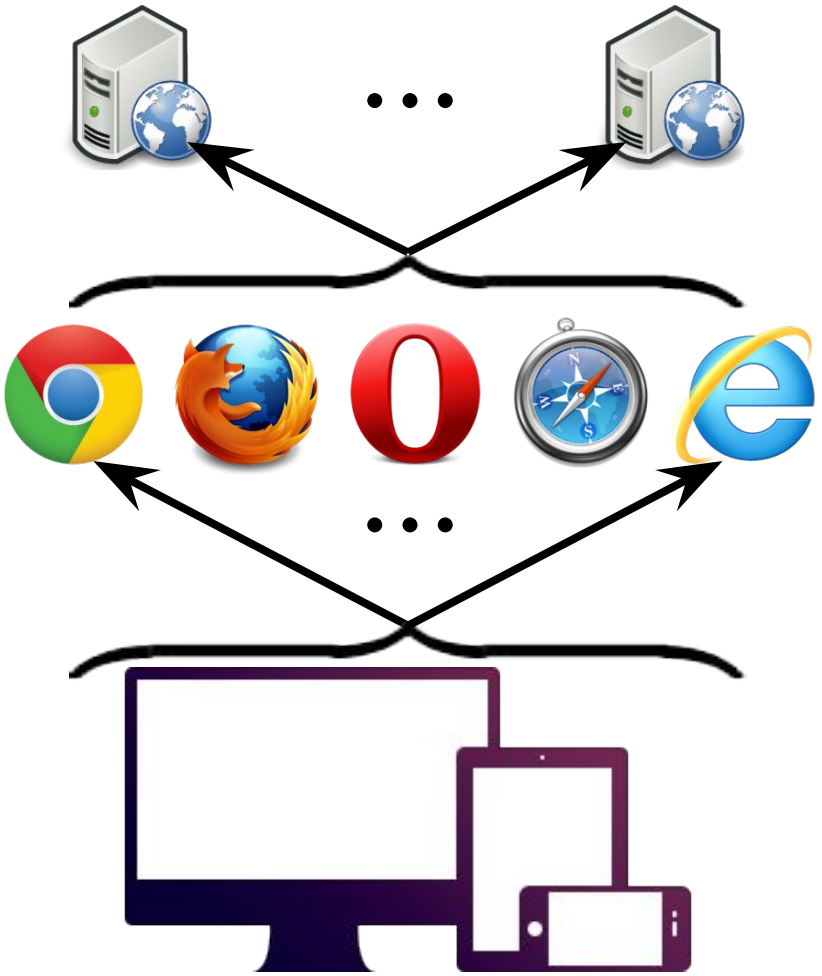
Dr. S.R. Shinde

Outline

- **Web Threats and Attacks**
- **Countermeasures**
- **Database Security**

Introduction

- Average user spends 16 h/month online (32 h/month in U.S.)
  - People spend much time interacting with Web, Web applications (apps)
  - Their (lack of) security has major impact
- Interaction via Web browser



Information Leakage

- Sensitive information can be leaked via Web:
  - All files accessible under a Web directory can be downloaded via GET requests
  - Example 1:
    - http://www.website.com/secret.jpg publicly accessible
    - http://www.website.com/index.html has no link to secret.jpg
    - Attacker can still download secret.jpg via GET request!
  - Example 2: searching online for “proprietary confidential” information

Misleading Websites

- Cybersquatters can register domain names similar to (trademarked) company, individual names
- Example: http://www.google.com vs. http://gogle.com vs. …
- Practice is illegal *if* done "in bad faith"
- Arbitration procedures available for name reassignment (ICANN)

XSS and CSRF

- Cross-site scripting (XSS): inject JavaScript from external source into insecure websites
  - Example: input `<script type="text/javascript"><!--evil code--></script>`
- Cross-site request forgery (CSRF): force victim browser to send request to external website → performs task on browser's behalf
  - Example: force load `<img src="http://www.bigbank.com/transferFunds.php?from=User&to=Attacker"/>`

SQL Injection

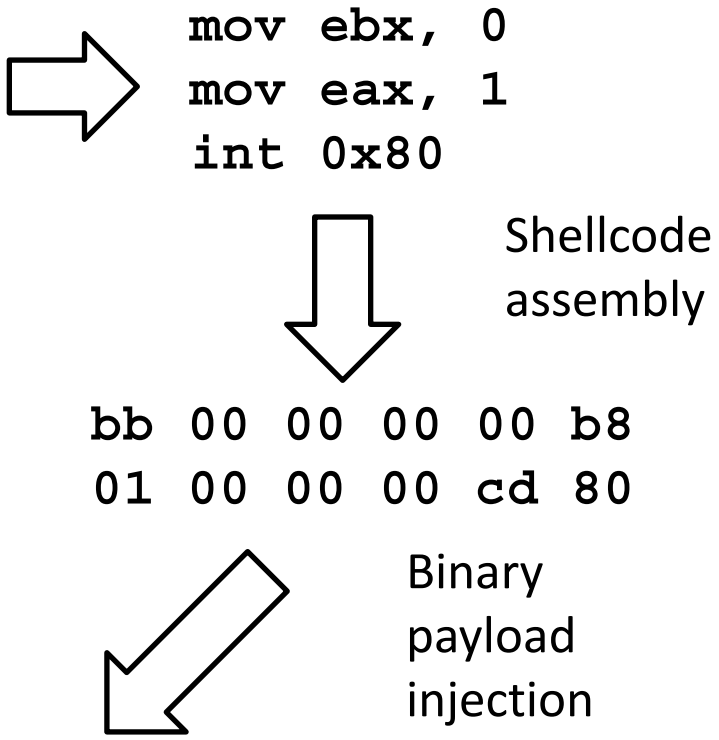
- Common vulnerability (~71 attacks/hour)
- Exploits Web apps that
  - Poorly validate user input for SQL string literal escape characters, e.g., `'`
- Example:
  - `"SELECT * FROM users WHERE name = '" + userName + "';"`
    - If `userName` is set to `' or '1'='1`, the resulting SQL is `SELECT * FROM users WHERE name = '' OR '1'='1';`
    - This evaluates to `SELECT * FROM users` ⇒ displays all users

Malicious Shellcode

- Shellcode is *non-self-contained* binary executable code
  - Distinct from malware that executes on its own
  - Shellcode can only execute after injection into a running process's virtual address space
- Most shellcode written in Intel IA-32 assembly language (x86)
- When injected into JS code, shellcode executes
  - Hijacks browser process
  - Can totally control target process or system
- Shellcode: attack vector for malicious code execution on target systems (e.g., Conficker worm)
  - Usually, browser downloads JS code containing shellcode
  - JS code executes, controls target process/system

A Toy Shellcode

- Shellcode for `exit()` system call
  - Store `0` into register `ebx`
  - Store `1` into register `eax`
  - Execute instruction `int 0x80`
- Assembled shellcode injected into JS code



JS code	`...3caabb00000000b801000000cd80ad46...`	more JS code

Disguised as normal data; injected into target processes' address spaces; compromises target processes' security

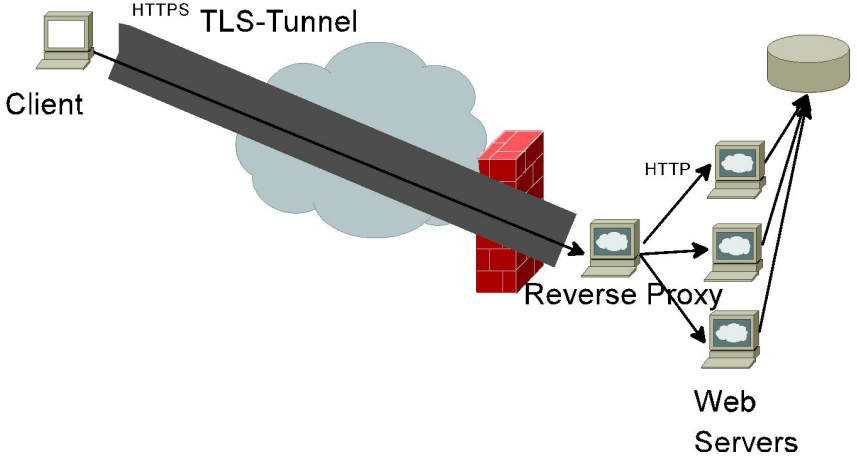
Countermeasures

HTTPS (HTTP Secure)

- HTTPS uses cryptography with HTTP
  - Alice, Bob have public, private keys; public keys accessible via certificate authority (CA)
  - Alice encrypts message with Bob's public key, signs message with her private key
  - Bob decrypts message with his private key, verifies message using Alice's public key
  - Once they "know" each other, they can communicate via symmetric crypto keys
- HTTPS provides greater assurance than HTTP

TLS/SSL

- HTTPS uses Transport Layer Security (TLS), Secure Sockets Layer (SSL), for secure data transport
  - Data transmitted via client-server “tunnel”
  - Much harder to compromise than HTTP
- Problems:
  - Relies on CA infrastructure integrity
  - Users can make mistakes (blindly click “OK”)



HTTPS Example

- User visits website via HTTPS, e.g., https://gmail.com
- Browser sends TLS/SSL request, public key, message authentication code (MAC) to gmail.com; gmail.com does likewise
  - TLS/SSL encrypt entire connection; HTTP layered atop it
  - Both parties verify each other's identity, generate symmetric key for following communications
- Browser retrieves public key certificate from gmail.com signed by certificate authority (Equifax)
  - Certificate attests to site's identity
  - If certificate is self-signed, browser shows warning
- Browser, gmail.com use symmetric key to encrypt/decrypt subsequent communications

Blacklist Filtering (1)

- Misleading websites: Register domain names similar trademarks, e.g., www.google.com, gogle.com, etc.
- XSS:
  - Validate user input; reject invalid input
  - Blacklist offending IP addresses
- CSRF:
  - Use random “token” in web app forms
  - If token is replayed, reject form (blacklist IP addresses)
- SQL injection:
  - Validate user input to databases, reject invalid input
  - Blacklist IP addresses

Blacklist Filtering (2)

- Helpful browser extensions:
  - NoScript/NotScripts/… (stop XSS)
  - AdBlock (can stop malicious scripts in ads)
  - SSL Everywhere (force HTTPS)
  - Google Safe Browsing
  - etc.

Defending Against Shellcode

- Two main detection approaches:
  - Content Analysis
    - Checks objects' contents before using them
    - Decodes content into instruction sequences, checks if malicious
  - Hijack Prevention
    - Focuses on preventing shellcode from being fully executed
    - Randomly inserts special bytes into objects' contents, raises exception if executed
    - Can be thwarted using several short "connected" shellcodes

Content Analysis

- Two major types of content analysis:
  - Static Analysis
    - Uses signatures, code patterns to check for malicious instructions
    - Advantage: Fast
    - Disadvantages: Incomplete; can be thwarted by obfuscation techniques
  - Dynamic Analysis
    - Detects a malicious instruction sequence by emulating its execution
    - Advantages: Resistant to obfuscation; more complete than static analysis
    - Disadvantage: Slower
- Focus on dynamic analysis (greater completeness)

Dynamic Analysis

- Approaches assume self-contained shellcodes
- Analyses' shellcode emulation:
  - Inefficiently uses JS code execution environment information
  - All memory reads/writes only go to emulated memory system
  - Detection uses GetPC code
- Current dynamic analysis approaches can be fooled:
  - Shellcode using JS code execution environment info
  - Shellcode using target process virtual memory info
  - Shellcode not using GetPC code
- To detect all malicious shellcodes, we need a better approach

Summary

- Web based on plaintext HTTP protocol (stateless)
- Web security threats include information leakage, misleading websites, and malicious code
- Countermeasures include HTTPS, blacklist filtering mechanisms, and malicious code detection

Database Security

- **Database security** concerns the use of a broad range of information security controls to protect databases (potentially including the data, the database applications or stored functions, the database systems, the database servers and the associated network links) against compromises of their confidentiality, integrity and availability. It involves various types or categories of controls, such as technical, procedural/administrative and physical.

Database Security Requirements

- Security requirements for databases and DBMSs:
  a. Physical database integrity requirements
     - DB immune to physical problems (e.g., power failure, flood)
  b. Logical database integrity requirements
     - DB structure preserved (e.g., update of a field doen't affect another)
  c. Element integrity requirements
     - Accuracy of values of elements
  d. Auditability requirements
     - Able to track who accessed (read, wrote) what
  e. Access control requirements
     - Restricts DB access (read, write) to legitinmate users
  f. User authentication requirements
     - Only authorized users can access DB
  g. Availability requirements
     - DB info available to all authorized users 24/7

Confident. / Integrity / Availability

- Requirements can be rephrased / sumarized as follows:
  - Data must be trusted
    - DBMS designed to manage trust
    - DBMS must reconstruct reality
  - Data must be accurate
    - Field checks
    - Access control (CRUD)
      - CRUD = **C**reate, **R**ead, **U**pdate, and **D**elete
    - Change log
  - Trade-offs
    - Audit vs. performance
    - Access vs. performance
  - Self-authentication
  - High availability

Reliability and Integrity

- Reliable software runs long time without failures
- Reliable DBMS preserves:
  - DB Integrity / Element Integrity / Element Accuracy
- Basic protection provided by OS underlying DBMS
  a) File back ups
  b) Access controls
  c) Integrity checks
- DBMS needs more CIA controls
  a) E.g. two-phase commit protocols for updates
  b) Redundancy/internal consistency controls
  c) DB recovery
  d) Concurrency/consistency control
  e) Monitors to enforce DB constraints
     - Range, state, transition constraints
     - Control structural DB integrity

Source: Pfleeger & Pfleeger

a) Two-Phase Update (2PC)

- Intent Phase
  - Check value of COMMIT-FLAG
  - Gathers resources
    - » Data
    - » Dummy records
    - » Open files
    - » Lock out others
    - » Calculate final answers
  - Write COMMIT-FLAG
- Permanent Change Phase
  - Update made
- Rollback ability at each phase

Detecting Inconsistencies

b) Redundancy/internal consistency controls
- Error detection / error correction
  - Hamming codes
  - Parity bits
  - Cyclic redundancy check
- Shadow fields

c) DB recovery
- Uses DBMS access log

d) Concurrency control
- Checks/enforcement

e) Monitors for DB constraints
- Range comparisons
- State constraints
- Transition constraints

} More sophisticated

- Sensitive Data

- Managing access
- Hiding existence
- Sharing vs. confidentiality
- Security vs. precision
  - Perfect confidentiality
  - Maximum precision

Source: Pfleeger & Pfleeger

Inference (Inference Problems)

- Inference attack - inferring *sensitive* data from *nonsensitive* data
- Types of inference attacks:
  1) Direct attack
     - Infer sens. data from results of queries run by attacker
     - *n*-item *k*-percent rule:
       - Data withheld if n items represent > k percent of the result reported
         - » Most obvious case: 1-item 100-percent case: 1 person represents 100 % of results reported
  2) Indirect attack
     - Infer sens. info from statistics (Sum, Count, Median) also from info external to the attacked DB
     - Tracker attacks (intersection of sets)
     - Linear system vulnerability
       - Use algebra of multiple equations to infer

Source: Pfleeger & Pfleeger

Inference Controls - Outline

1) Query controls — applied to queries
   - Primarily against direct attacks
   - Query analysis to prevent inferences
   - Query inventory (history) per person
2) Data item controls —applied to individual DB items
   - Useful for indirect attacks
   - Two types:
     a) Suppression — data not provided to querying user
        - Suppress combinations of rows and columns
        - Combine results (to hide actual answers)

     b) Concealing — close answers, not exact given to querying user
        - Rounding
        - Present range of results
        - Present random sample results
        - Perturb random data (generate small + and – error)

Source: Pfleeger & Pfleeger

Database Inference Problem & Types

- DB inference problem:

Non-sensitive information + Meta-data = Sensitive information

where meta-data:

  - Working knowledge about the attributes
  - Supplementary knowledge (not stored in database)

- DB inference types:

  1) Statistical database inferences
  2) General-purpose database inferences

1) Statistical Database Inference

- Statistical database goal: provide aggregate information about groups of individuals
  - E.g., average grade point of students
- Security risk in statistical database:

  disclosure of specific information about a particular individual
  - E.g., grade point of student John Smith

Types of Statistics

- Macro-statistics: collections of related statistics presented in 2-dimensional tables

Sex\Year	1997	1998	Sum
Female	4	1	5
Male	6	13	19
Sum	10	14	24

- Micro-statistics: Individual data records used for statistics after identifying information is removed

Sex	Course	GPA	Year
F	CSCE 590	3.5	2000
M	CSCE 590	3.0	2000
F	CSCE 790	4.0	2001

Statistical Compromise

- Exact compromise:

  Find exact value of an attribute of an individual

  – E.g., finding that John Smith's CGPA is 3.8

- Partial compromise:

  Find an estimate of an attribute value corresponding to an individual

  – E.g., finding that John Smith's CGPA is between 3.5 and 4.0)

Methods of Attacks and Protection

- Small/Large Query Set Attack
  - C: characteristic formula that identifies groups of individuals

  If C identifies a single individual I, e.g., count(C) = 1

  - Find out existence of another property D for I
    - If count(C and D)= 1 means I has property D
    - If count(C and D)= 0 means I does not have D

  OR

  - Find value of property
    - Sum(C, D), gives value of D
      - If value of C known already

Prevention

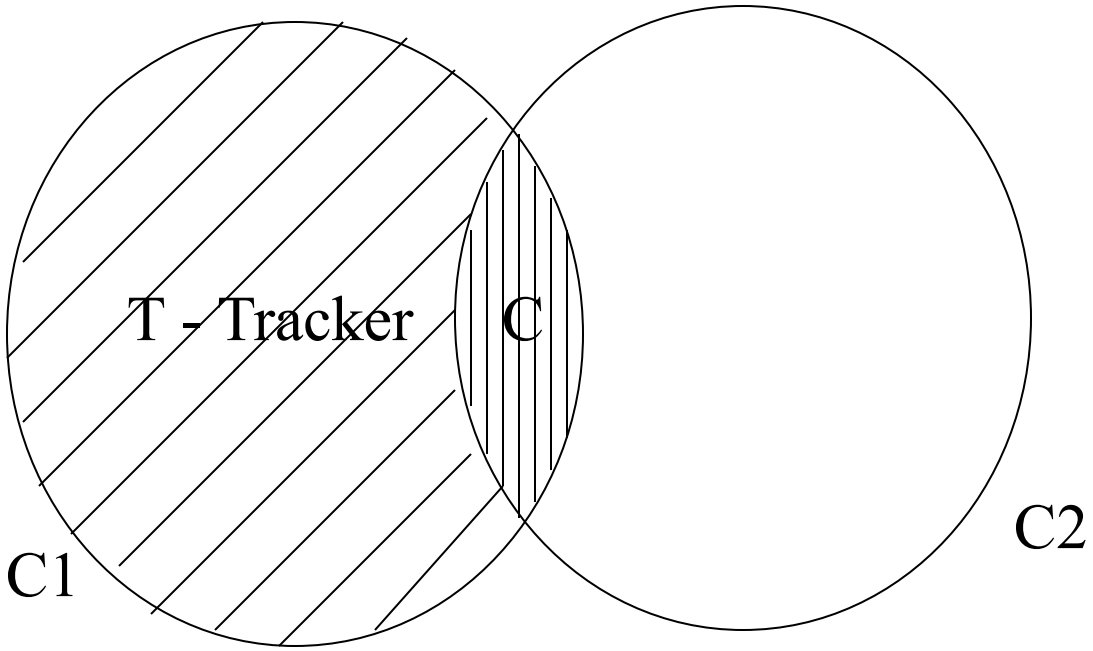
- Protection from small/large query set attack:
  query-set-size control

- A query q(C) is permitted only if
  $N-n \geq |C| \geq n$
  where:
  $n \geq 0$ is a parameter of the database, and
  N is the number of records in the database

- E.g. a query q(C) in a DB describing 100 individuals is permitted only if
  $100 - 5 = 95 \geq |C| \geq 5$
  that is if it can't give statistics on a group smaller than 5 individuals
  (Note: If it gives statistics on C for e.g., 96 people, it gives statistics on not-C for 4 people.)

Tracker Attack 1 (simple)

Query q(C) is disallowed



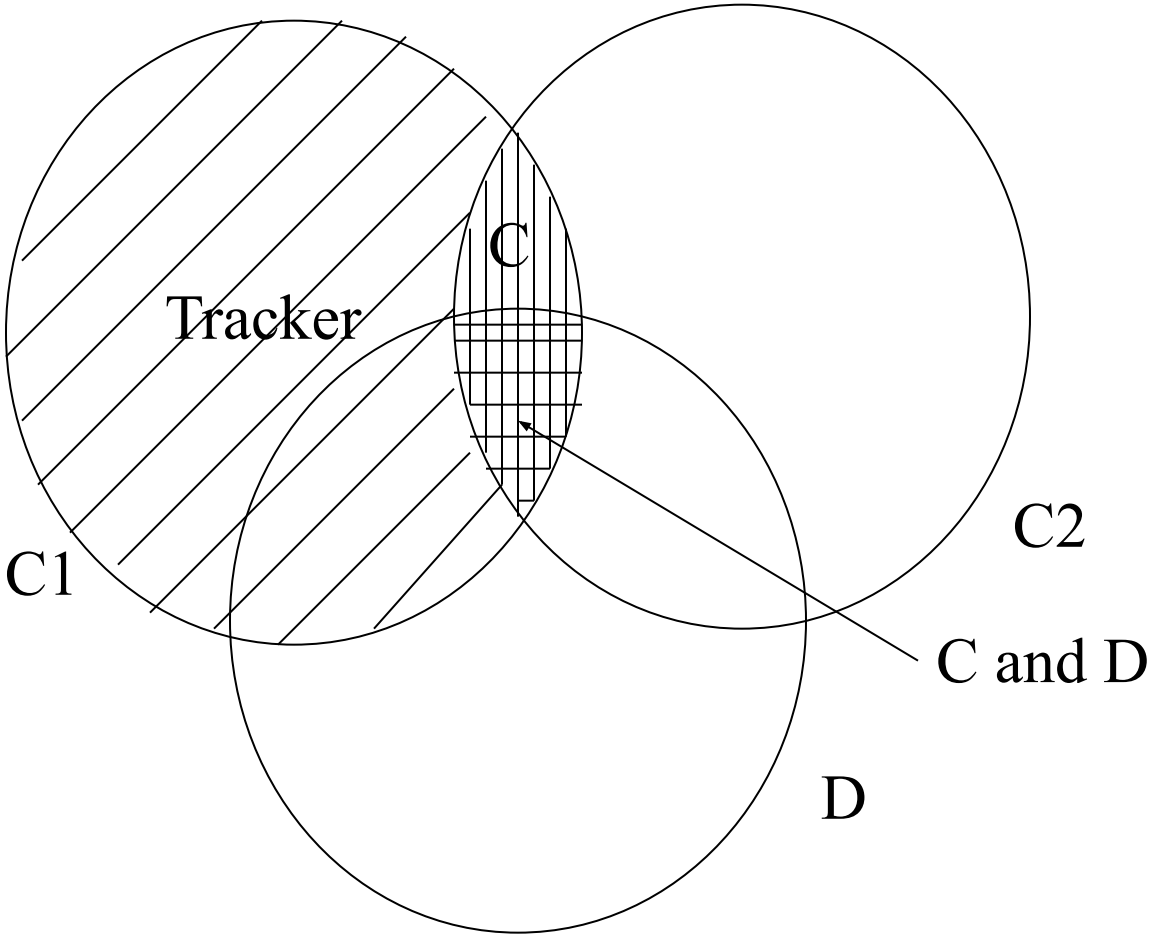
C = C1 and C2
T = C1 and ~C2

Attacker runs instead 2 queries: q(C1) and q(T)
where q(C) = q(C1) – q(T)
=> infers q(C) from q(C1) and q(T)

Tracker Attack 2 (more complex)

Query q(C and D)
is disallowed

C = C1 and C2
T = C1 and ~C2

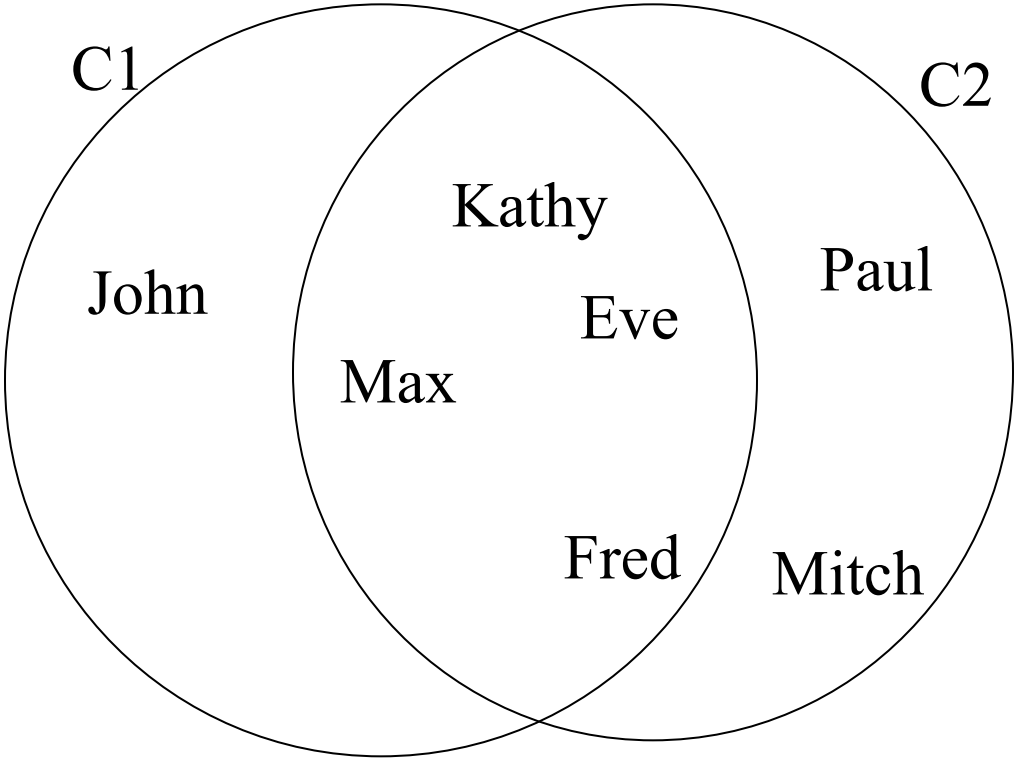


Attacker runs instead 2 queries: q(T or C and D) and q(T)
where q(C and D) = q(T or C and D) – q(T)

=> infers q(C and D) from q(T or C and D) and q(T)

Query overlap attack

Q(John)=q(C1)-q(C2)



Protection: need query-overlap control

Insertion/Deletion Attack

- Observing changes over time
  - q_1=q(C)
  - Insert(i)
  - q_2=q(C)
  - q(i) = q_2 „-” q_1
    - where „-” means compensation for insertion that permist to infer

- Protection: insertion/deletion performed as pairs

Statistical Inference Theory

- Given unlimited number of statistics and correct statistical answers, all statistical databases can be compromised

  [Ullman]

- Fortunately:
  - Number of statistics can be limited by statistical DB controls
  - Statistical DB can give approximate rather than 'correct' statistical answers

2) Inferences in General-Purpose Databases

- Inference types:
  a) Inference via queries based on sensitive data
  b) Inference via DB constraints
  c) Inference via updates

a) Inference via Queries Based on Sensitive Data

- Sensitive information is used in selection condition but not returned to the user
- Example: Salary: secret, Name: public

$\pi_{Name}\sigma_{Salary=\$25,000}$ (π- projection, σ - selection)

$\pi_{Name}\sigma_{Salary=\$26,000}$

• • •

$\pi_{Name}\sigma_{Salary=\$110,000}$

  - Sensitive info (salary) used in selection condition, but not returned to the user
    - Returns only Name to user
  - "Infers" (quite mechanically – no intelligence needed) salary for everybody making between $25,000 and $110,000

- Protection: apply query of database views at different security levels

b) Inference via DB Constraints

- Database constraints:
  - b-1) Integrity constraints
  - b-2) DB dependencies
  - b-3) Key integrity

b-1) Infering via Integrity Constraints

- C = A+B
- A - public, C - public, and B - secret
- B can be calculated from A and C
  - I.e., secret information can be calculated from public data

b-2) Infering via DB Dependencies

- DB dependencies (metadata):
  - Functional dependencies
  - Multi-valued dependencies
  - Join dependencies
  - etc.

Functional Dependencies

- Functional dependency (FD) for attributes A □ B:

  For any two tuples in the relation, if they have the same value for A, they must have the same value for B

- Example: Exploiting the FD: Rank □ Salary to infer secret info

  Secret information: Name and Salary together

  – Query1: Name and Rank
  – Query2: Rank and Salary
  – Combined answers for Q1 and Q2 reveal Name and Salary together
    - Only because we have Rank □ Salary

b-3) Infering via Key Integrity

- Every tuple in the relation has a unique key
- Users at different security levels see different versions of the database
  - User with ‘top secret’ clearance sees more than one with ‘secret’ clearance
- Users might attempt to update data that is not visible for them

Example – Infering via Key Integrity

Secret View

Name (key)	**Salary**	**Address**
Black P	38,000 P	Columbia S
Red S	42,000 S	Irmo S

Public View

Name (key)	**Salary**	**Address**
Black P	38,000 P	Null P

Example (ctd) - Updates

Public User:

Name (key)	Salary	Address
Black P	38,000 P	Null P

1. Update Black's address to Orlando
2. Add new tuple: (Red, 22,000, Manassas)

If

Refuse update => covert channel

Allow update =>

- Overwrite high data – may be incorrect
- Create new tuple – which data it correct (polyinstantiation) – violate key constraints

  polyinstantiation – given record instantiated many times, each time with different security level

Example (ctd) - Updates

Secret user:

Name (key)	Salary	Address
Black P	38,000 P	Columbia S
Red S	42,000 S	Irmo S

1. Update Black's salary to 45,000

If

Refuse update => denial of service

Allow update =>

- Overwrite 'low' data – covert channel
- Create new tuple – which data it corrects
  (polyinstantiation) – violate key constraints

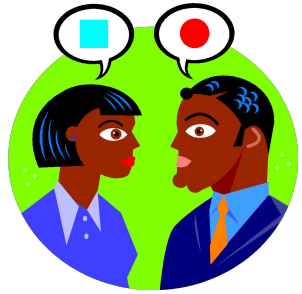
  polyinstantiation – given record instantiated many times, each time with different security level

Conclusions on Inference

- No general technique is available to solve the inference problems
- Need assurance of protection
- Hard to incorporate outside knowledge
- Optimal plan:
  - Suppress obviously sensitive information
  - Track what user knows (expensive)
  - Disguise data
- - Aggregation—additional problem
  - Inferences from aggregating data
  - Data mining increases risks

Source: Pfleeger & Pfleeger

Multilevel Databases



- Multilevel databases - store data with different sensitivity levels (e.g.: public, confidential, secret, top_secret)

- Problems
  - Polyinstantiation – multiple ("poly") instantiations of a record, each at a different security level
    - Example:
      - [John, Kalamazoo-MI] -- Public level
      - [John, 19_Main_Ave-Kalamazoo-MI] -- Confidential level

        …
      - [John, 19_Main_Ave-Kalamazoo-MI, …, SSN=123-45-6789] -- Top_Secret level

Source: Pfleeger & Pfleeger

Proposals for Multilevel Security - Separation Mechanisms

1) Partitioning
   - Redundancy
   - Accuracy (multiple field update)
2) Encryption per level
   - Cumbersome decrypting with queries
3) Integrity lock
   - Data item
   - Sensitivity level
   - Checksum (above 2)
   - Cryptographic checksums
4) Sensitivity lock
   - Unique identifier
   - Sensitivity level

Source: Pfleeger & Pfleeger

Implementations of Separation - 1

1) Integrity lock
   - Expands size of element
   - Processing time efficiency
   - Untrusted DBM subject to Trojan horse

2) Trusted front end
   - Guard ~ reference monitor
   - One-way filter—filters out reports
   - Inefficient—calls, then releases much data

3) Commutative filters
   - Interface between user and DB
   - Reformats query
   - Addresses inefficiencies (above)

Source: Pfleeger & Pfleeger

Implementations of Separation - 2

4) Distributed DB
   - Separate DB's based on sensitivity
   - Front end sends query to right DB
5) Views
   - Logical / functional divisions

Source: Pfleeger & Pfleeger