

Database Security Issues

Victor W. Marek

Computer Science Department

University of Kentucky

Lexington, KY

Fall 2007

- ▶ Motivations
- ▶ Attacks on databases
 - Trojan horse attack
 - SQL Insertion attack
- ▶ Access Control Models
 - “Chinese Wall” model
 - Decentralized control (DAC; SQL)
 - Centralized control (MAC; Bell-LaPadula, Biba models)
- ▶ Testing security of DBMSes
- ▶ Conclusions

- ▶ Access to information unprecedented in the history of humanity
- ▶ Information stored in databases
- ▶ Sensitivity of information, and benefits (sometimes ill-gained) from the access to such information
- ▶ Databases as a back-end of commerce, in particular e-commerce
- ▶ Databases in medical care
- ▶ Databases in the office
- ▶ Databases in financial industry
- ▶ Databases in national security (non-fly list)

- ▶ Trade in secret information
 - Semi-legal trade in sensitive information (just go to 411.com or intelius.com and see for yourself)
 - Alleged trade in very sensitive financial information
- ▶ Local excesses
 - Until recently the use of SS numbers of students as student IDs
 - No reasonable protection of student data
 - Faculty still identified by SS numbers in the University information system

- ▶ Clooney Incident
 - When the actor George Clooney, had a motorcycle accident, television stations in Los Angeles were broadcasting within an hour the medical problems suffered by the actor and his companion
 - Subsequent investigation found that during that one hour forty (40!) individuals on the staff of the hospital made unauthorized access to Mr. Clooney medical record. **This number did not include individuals with *need to know***
- ▶ It should be clear that preventing *social engineering* behavior like this one, or in general *Kevin Mitnick*-like behavior, is beyond CS, similarly like making people lock the door is not an enforceable
- ▶ Nevertheless *some* aspects of *social engineering* can be prevented by CS and hence should be eliminated via software

► TJ Maxx Incident

- Allegedly, starting in January 2005, using a parabolic antenna and a computer, four Florida criminals intercepted the transfer of data between cash registers and company databases, getting an access to a large amount of credit card information
- Allegedly, the store that was hacked was in Minneapolis, MN
- It appears that a weak, easily breakable encryption (WEP protocol) was used

- ▶ Economical and societal consequences
 - TJ Maxx is a publicly traded company, with several billions of dollars in sales, and several lines of stores, such as *HomeGoods*, *TJ Maxx*, and *Marshall's*
 - It was reported that 45.7 millions credit card numbers have been revealed (but recent court filings alleged 96 millions) and the loss for the Visa CC company alone was *estimated* at \$68M to \$83M
 - The incident was not a single event, but lasted, reportedly, several months
 - The matter is litigated right now

- ▶ Security is defined as:
 - (Availability) Access to information of authorized users
 - (Secrecy) No access to information to unauthorized users
 - (Integrity) Only authorized users should be able to modify data

Attacks on databases, “Trojan horse”

- ▶ Two users of databases: *marek* and *mirek*. The user *marek* is a “bad guy”, while the user *mirek* possesses an information that the user *marek* wants. The bad guy *marek* wants this information on the continuing basis
- ▶ The bad guy *marek* does not want to use *cron* because the administrator *raphael* will see it at once
- ▶ The user *marek* breaks into *mirek*’s account once and does three things. First, installs a hidden application program. Second, using the *active component* of DBMS installs a rule that executes application program (which copies the user *mirek* updates into a table in *marek*’s table.) Third, gives user *mirek* permission to write to *marek*’s table.

Attacks on databases, “Trojan horse”, cont’d

- ▶ (Of course the break is executed this way: *marek* breaks into *mirek*’s office. *mirek* is not in habit of locking his screen as he leaves the office. *marek* securely copies from some external untraceable account (in *Honduras*) the application program and installs the rule in the active component of *mirek*’s db)
- ▶ The bad guy *marek*, in the secrecy of his office gives *mirek* permissions to write to *marek*’s table
- ▶ What kind of security model enables such attack?

- ▶ The owner *mirek* operates a small but profitable e-commerce jewelry store *Jewelry R Us*, a.k.a. *jerus.com*. The bad guy *marek* intends to extort from *mirek* a significant amount of money. He is, actually, a former employee of *FBN Operations*^a, the company that maintains the e-store.
- ▶ The threat sent in by *marek* (from *Honduras*, of course) is “If you do not put under the stone in front of your shop an envelope with \$10,000.00 (in \$100.00 notes), I will make your business suffer”
- ▶ *mirek* contacts the police which puts the stone under observation

^aAlso known as Fly-by-Night Operations

SQL Insertion attack, cont'd

- ▶ *marek* who previously put the stone (and hence the police) under observation, needs to let *mirek* know that he (*marek*) is not a “paper tiger”
- ▶ The access to the store database is protected by password. The password is checked against contents of a database. In the suitable box *marek* (as usual from *Honduras*) writes

";DROP TABLE clients; – ; "DROP TABLE goods; –

SQL Insertion attack, cont'd

- ▶ Since the owner *mirek* hired as implementators a company FBN Operations, their implementation of the e-store did not validate input
- ▶ The effect was that the tables *clients* and *goods* were erased and the store was, effectively, closed
- ▶ The owner *mirek* lost his business for the next 16 hours (until the backups are retrieved), lost a number of clients etc.
- ▶ (This scenario was used in extortions from off-shore casinos)
- ▶ (A suitable care (putting user input in quotes) prevents SQL insert attack)

- ▶ The attacks described above are *similar* to many attacks on systems
- ▶ SQL injection, for instance, is similar to *buffer overflow* attack
- ▶ But each one had a *database flavor* related to the way databases do business
- ▶ Each one exploited a different, database-related vulnerability
- ▶ There may (and will) be more vulnerabilities and so, when I hear about *Microsoft* initiative for *Central depository of medical records*, it gives me creeps

Models of access control

- ▶ Access to data must be controlled
- ▶ Many models of access control proposed in the literature
- ▶ Operating systems area faces similar issues, so literature overlaps
- ▶ In 1970ies and 1980 US DoD supported many projects in this area
- ▶ We will discuss three models
 - “Chinese Wall”
 - Discretionary Access Control
 - Mandatory Access Control (actually two models, there are many)
- ▶ The reason is that these are in use currently

- ▶ The simplest of them all
- ▶ We create a physical barrier between databases used by the group of authorized users and the rest
- ▶ Widely used in financial institutions, for instance in brokerage houses that both manage other people money and invest themselves
- ▶ The issue is that the analysts while researching companies learn significant information, often *not publicly available*

- ▶ The buying/selling arm of the brokerage can not have access to such information for it violates level field principles
- ▶ Breaches occur often, viz. recent incident at *Mizuho Corporate Bank* and *Mizuho Securities* reported on October 22 in *Wall Street Journal*
- ▶ In principle, complete isolation of a database system prevents *Trojan horse*

Access control models, cont'd

- ▶ Who controls the access to database tables?
- ▶ Two basic ideas
 - Decentralized
 - Centralized
- ▶ The current most popular access control model is *Discretionary Access Control* model of SQL
- ▶ But current events such as Sarbanes-Oxley Act force some businesses to use versions of a centralized model, called *Mandatory Access Control*
- ▶ The idea: the access to database objects (such as tables and attributes) requires permissions

Discretionary Access Control

- ▶ Types of access: *insert, delete, select, update*
- ▶ Permissions can be *granted* and *revoked*
- ▶ When a table is created, its owner has all permissions
- ▶ An owner can *grant* permissions with two levels
 - Simple permissions
 - *with grant option* permissions
- ▶ These last permissions allow the *grantee* to further pass permissions

- ▶ Let us look at an example

The owner of jewelry store, *mirek* gives his employee *marek* permission to view content of *inventory* table:

```
GRANT UPDATE TO marek ON inventory
```

And here is the stronger version:

```
GRANT UPDATE TO marek ON inventory WITH GRANT  
OPTION
```

- ▶ Later on, *mirek* realizes his mistake and revokes the permission
REVOKE UPDATE FROM marek ON inventory CASCADE
- ▶ This last qualifier CASCADE revokes the permissions from *marek's*
grantees
- ▶ In fact, SQL supports the check (to eliminate unexpected results) of
further permissions (via an option called RESTRICT)

Discretionary Access Control, cont'd

- ▶ Once the owner *mirek* gave *marek* permissions WITH GRANT OPTION he lost the control over who will have the access to *inventory* table
- ▶ For bad guy *marek* may give the access to all sort of users
- ▶ But DAC is convenient. For instance the owner *mirek* can delegate various tasks periodically without problems to other users and then revoke permissions
- ▶ DAC is not difficult to explain and excellent for fast paced world of business
- ▶ What is the semantics of DAC, though?

Discretionary Access Control, cont'd

- ▶ The semantics of DAC is implemented by a labeled graph (to be precise a multigraph)
- ▶ The nodes are user names
- ▶ The labels are permissions
- ▶ There is an additional node called *system*

- ▶ The system grants an appropriate permission if there is a path from the system to the requesting user with the requested access mode, and with all labels, with the possible exception of the last one, having grant option
- ▶ The owner *mirek* gives permission to select on *inventory* to *marek* and the user *marek* gives select permission to *tony*. Then the select access for *tony* is NOT granted
- ▶ The owner *mirek* gives permission to select on *inventory* to *marek* with grant option and the user *marek* gives select permission to *tony*. Then the select access for *tony* is granted
- ▶ Since multiple paths from *system* node to a given node may exist, this semantics has possibly unintuitive consequences

- ▶ But DAC is very convenient, and SQL supports it
- ▶ Clearly, DAC is subject to the *Trojan Horse* attack
- ▶ While it is possible to modify DAC to deal with that attack, for instance by maintaining *access lists*, we are not aware of commercial systems that have this feature

- ▶ Security in SQL is provided by three concepts:
 - Discretionary Access Control
 - Views: limiting access to columns
 - Roles: Creating groups of users with equal access rights

- ▶ There are various models of security in databases that support *centralized* or *mandatory* access control (clearly, DAC was *decentralized*)
- ▶ We will discuss two such models (but there are many)
 - Bell-LaPadula model (that is oriented towards *secrecy*)
 - Biba model (that is oriented towards *data integrity*)
- ▶ Both are based on a dichotomy *users/objects*

- ▶ There is a finite collection C of *categories*
- ▶ There is a collection H of classifications (in BLP model, it is assumed it has four classifications: U, C, S, TS – but there may be different classifications)
- ▶ There is an ordering \leq_H on classifications
- ▶ In BLP it is a linear ordering, but does not have to be
- ▶ Subjects are users, their application programs, etc.

- ▶ *Subjects* are provided with an array of *clearances*, one for each category. We denote $f(S)$ its clearance (i.e. the array)
- ▶ It is assumed (but there are variations) that application programs have same clearances as their creators
- ▶ Objects are provided with an array of *classifications*. We denote $f(O)$ its array of classifications
- ▶ The ordering \leq_H implies a (partial) ordering of clearances - the *product ordering*
- ▶ Someone (called *security officer*) assigns classifications to objects. Usually it has something to do with the subject that creates them
- ▶ In principle, MAC distinguishes between *database administrators* (DBA) and security officers

- ▶ Here the issue is *secrecy*
- ▶ The semantics is provided by two major rules of gaining access plus one more minor rule

- Subject S can *write* into object O if

$$f(S) \leq f(O)$$

- Subject S can *read* into object O if

$$f(S) \geq f(O)$$

- ▶ The first of these these rules protects confidentiality of information
- ▶ The second of these these rules protects availability of information
- ▶ Then, there is a rule called *tranquility* rule, namely that when an object is processed, its classification does not change (so we can't hack it)
- ▶ Apparently, BLP model is driven by *need to know*

Bell-LaPadula model, cont'd

- ▶ There are all sorts of variations (for instance may be we can separately consider *inserts* and *deletes*)
- ▶ Clearly, BLP can be used to protect against Trojan Horse scenario
- ▶ To the best of my knowledge, SQL systems do not provide BLP security, but at least to some extent UNIX systems do provide such implementations
- ▶ SQL:1999 does not provide any standard for BLP

- ▶ Here the issue is not secrecy, but rather *integrity* of data
- ▶ There is no tranquility; the level of the object may adjust
- ▶ The rules for access are *opposite* to BLP rules
 - Subject S can *write* into object O if

$$f(S) \geq f(O)$$

- Subject S can *read* into object O if

$$f(S) \leq f(O)$$

- ▶ The idea is that, if your information is more correct (maybe...) so the document is more important
- ▶ Biba clearances reflect position in the hierarchy and subjects with higher clearances have better knowledge (?) so produce more precise data

- ▶ There is no support for Biba model in SQL standard
- ▶ I believe that unlike BLP (that is mentioned in every security book), Biba model has a limited application
- ▶ While there are many models in the literature, the ones I discussed dominate

- ▶ The issue of testing *systems*, and in particular *database systems* is present since the inception of computers
- ▶ The security specification for *trusted computer systems*, so-called *Orange Book* has been published by US Department of Defense in 1985
- ▶ It was superseded by so-called *Common Criteria*, a joint initiative of several European countries and US. These are often called *EU-US Common Criteria* and are maintained by a number of organizations in Europe and in US
- ▶ CC are also adopted by International Standards Organization as ISO 15048

- ▶ The issue of testing is two-fold:
 - How the products are tested
 - Who can test

- ▶ Common Criteria introduce terminology and methodology of evaluation
 - *Targets of Evaluation* (what is evaluated)
 - *Protection Profiles* (there is a 3 volume (not kidding!) Protection Profile for Databases, explaining what are the issues in evaluating databases)

- ▶ More concepts
 - *Evaluation Protection Levels* (classifying degrees of security)
 - There are seven EALs, only four are valid internationally (but for those lower levels, certification from Estonia is valid in US!)
 - Independent, certified laboratories are supposed to certify
 - Governmental bodies provide final approval (needed for Government organizations to buy the product)

- ▶ But all these Common Criteria and the EALs have to be taken with a “grain of salt”
 - There are no penalties
 - To give an example: a major software company was able to get certifications of EAL4 for their operating systems and DBMSes but there was a number of well-published and successful attacks on their DBMS

Conclusions, message to take home

- ▶ Database Management System share security problems with other computer systems
- ▶ Due to the fact that DBMSes provide the back-end support for e-commerce, the security problems of databases have major economic impact
- ▶ There is plenty of work done in the area of DBMS security
- ▶ We are not defenseless in our DBMSes, but have to be vigilant
- ▶ Computer *Science* will not be able to solve all the problems of database security

- ▶ But, for instance, MAC can prevent *some* problems (e.g. *Trojan Horse*), while enforcing guidelines describing input processing can prevent SQL injection
- ▶ Our program devotes limited attention to database security; one of our lectures (Second database course) devotes a couple of weeks to DB security
- ▶ Clearly, users of databases, esp. over the WWW, prefer beautiful GUIs to mundane problems such as security
- ▶ All too often users and owners of software take security for granted
- ▶ What can be done by our discipline to make databases more secure - remains to be seen

Sources, further reading (and viewing)

- ▶ We list a number of sources on DB security
 - R. Ramakrishnan and J. Gehrke *Database Management Systems*, Chapter 21 (outdated)
 - R. Ben Natan *Implementing Database Security and Auditing*
 - D. Lichtenfield, C. Anley, J. Heaseman, and B. Grindlay *The Database Hacker's Handbook*
 - S. Castano, M. Fugini, G. Martella and P. Samarati *Database Security* - old but plenty of information on older work
 - N. Daswany, *What every engineer needs to know about security*, a presentation from *Google TechTalk*