CSIT115 Data Management and Security

Data Vulnerabilities

Dr Janusz R. Getta

School of Computing and Information Technology -
University of Wollongong

Data Vulnerabilities

Outline

Concepts

Risk

- Risk is the potential of gaining or losing something of value

Threat

- A threat is a communicated intent to inflict harm or loss on another person

Vulnerability

- Vulnerability refers to the inability (of a system or a unit) to withstand the effects of a hostile environment

Attack vector

- An attack vector is a path or means by which a hacker can gain access to a computer or network server in order to deliver a payload or malicious outcome
- Attack vectors enable hackers to exploit system vulnerabilities, including the human element

Data Vulnerabilities

Outline

What causes vulnerability ?

Software defects

- Software defects are accidentally or intentionally built into the code during software development and include design flaws and coding mistakes (35% of successful attacks exploit these types of errors)
- Design flaws involve design decision that create an inherently insecure system
- Coding errors include both ordinary software bugs as well as features that were put in not by design but through oversight (and as a result of developers not thinking of all potential consequences)
- Coding errors include buffer overflows, race conditions, back doors into systems, and even nonrandom random-number generators

What causes vulnerability ?

Configuration errors

- Configuration errors account for 65% of vulnerabilities
- Configuration errors include set up of unnecessary and dangerous services when a system is configured such that it brings up services and allows for connections that are not required
- It is usually caused by installation of a system with a default configuration rather than with precisely defined configuration that eliminates all features that are not required (it is easier to use default configuration because vendors prefer to offer an all-enabling starting configuration)

Access administration errors

- When access control includes configuration errors, entire security model falls apart
- Because most complex systems have elaborate access control schemes based on the concepts of groups, roles, permissions, delegation, etc it is easy to get the errors in access control configuration
- It is very hard to detect the cases that exploit such errors because it cannot be detected by intrusion detection or other monitoring systems due to incorrect assumptions that outside access looks correct

Data Vulnerabilities

Outline

The famous SQL Injection attack

SQL injection is a technique that exploits the applications using relational databases as their back end

The technique uses the fact that applications have an available connection to a database and that the application composes SQL statements and send them to a database server to extract data or to perform certain functions

SQL injection uses a fact that many of these applications compose such SQL statements by doing string concatenation of the fixed part of SQL statements along with user supplied data that forms `WHERE` clause or additional subqueries

The technique is based on intentionally malformed user-supplied data that transform SQL statement from an innocent form into a malicious call that causes unauthorized access, deletion of data, or even theft of information

In all cases SQL injection as a technique is based on using bugs and vulnerabilities in an application

The famous SQL Injection attack

For example, assume that we would like to implement the following interface that can be used to login a user to a system

```
USER ID:  ______________________________
PASSWORD: ______________________________
          +----------+    +----------+
          | Logon    |    | Enroll   |
          +----------+    +----------+
```

User interface

The application receives `USER ID` and `PASSWORD` and it authenticates by checking `USER ID` and `PASSWORD` in `USER` table

Additionally the application does not validate what a user typed into these two fields and SQL statement is created by string concatenation

The famous SQL Injection attack

The following piece of code implements the authentication

Embedded SQL

```
sqlString = "SELECT USERID FROM USER WHERE USERID = ' " &
userID& " ' AND PWD = ' " &pwd& " ' ";
result = GetQuery Result(sqlString);
if (result = "") then
        userHasBeenAuthenticated = False
else
        userHasBeenAutheticated = True
end if;
```

What happens when a user intentionally types in a malicious code like

Data entry

```
USER ID: ' OR ' ' = '
PASSWORD: ' OR ' = '
```

In such a case `sqlString` variable obtains the following value

SELECT statement

```
SELECT USERID FROM USER WHERE USERID = '' OR '' = '' AND PWD = '' OR '' = ''
```

The famous SQL Injection attack

In such a case `sqlString` variable obtains the following value

```
SELECT statement
SELECT USERID FROM USER WHERE USERID = '' OR ''= '' AND PWD = '' OR ''=''
```

Interpretation of `WHERE` condition returns `TRUE` because empty string is equal to empty string `(' ' = ' ')` and evaluation of disjunctions `USERID = ' ' OR ' '= ' '`, and `PWD = ' ' OR ' ' = ' '` returns `TRUE` and finally evaluation of conjunction `USERID = ' ' OR ' ' = ' ' and PWD = ' ' OR ' ' = ' '` returns `TRUE`

Hence `result` is not empty and a variable `userHasBeenAutheticated` is set to `True`

Data Vulnerabilities

Outline

Trojan

A Trojan is an unauthorized program contained within a legitimate program

A legitimate program is modified by placement of unauthorized code with it

A legitimate program seems to do one thing but it actually does several other operations without your knowledge or agreement

The word “Trojan” comes from the myth about the City of Troy in which the Greeks gave their enemy a “gift” – huge wooden horse as a gift during a war

The Greek soldiers were hidden inside the horse

The soldiers stormed out of the horse during the night and they conquered the City of Troy

Trojans (or Trojan horses) are one of the main forms of attack that have gained “fame” on the desktop computers together with worms, viruses, and other malicious software

Trojan

A Database Trojan is an attack that consists of two phases: the injection of the malicious code and the calling of the malicious code

It is difficult to track Database Trojan because of separation in two phases, it is difficult to associate two apparently not related events

A Database Trojan after it is inserted into the system may stay in the system for a long time ("sleeper") until it is activated

Trojan

There are four categories of Trojan attacks:

- An attack that both injects a Trojan and calls it
- An attack that uses an oblivious user or process to inject a Trojan and then calls it to extract the information or perform an action within a database
- An attack that injects a Trojan and then uses an oblivious user or process to call a Trojan
- An attack that uses an oblivious user or process to inject a Trojan and also uses an oblivious user or process to call a Trojan

An example of using an oblivious user is a scenario when a junior developer gest some procedural code (e.g. trigger or stored procedure) from someone he/she does not know and then uses this code without fully understanding what it is doing.

Data Vulnerabilities

Outline

Elimination of vulnerabilities

Track processing of stored procedures

- Create baseline for a set of stored procedure
- Monitor all divergences from a baseline
- Log information and analyse the logs
- Implement a real-time alert
- Implement base-line capable firewall

Control creation of and changes to procedures and triggers

Watch for changes to run-as privileges

Closely monitor developer activity on production environments

Monitor creation of traces and event monitors

Be aware of SQL attachments in e-mails

Data Vulnerabilities

Outline

Hardening MySQL environment

Physically secure server on which MySQL lives

Use the following values of system variables

- `local_infile = 'OFF'` to disable `LOCAL` in `LOADDATA` statements
- `skip_show_database= 'OFF'` to ensure that `show databases` command only lists databases for which the user has some kind of privilege; in a more restrictive approach use `skip-show-databases` option
- `secure_auth= 'ON'` to disallow authentication for accounts that have password from earlier versions
- `skip-name-resolve='ON'` Do not resolve host names when checking client connections and use only IP addresses

Do not grant `PROCESS, FILE`, or `SUPER` privileges to non-administrative users

Do not run MySQL server on the same host as Web server in order to force remote connections

Ensure strong password for a user `root`

Hardening MySQL environment

- Disallow the default full control of the database to local users and disallow the default permissions for remote users to connect to a database
- Do not use MySQL prior to version 4.1
- Limit privileges to the `load_file` function
- Disallow developers to access production database servers
- Enable auditing

References

R. Ben Natan, Implementing database security and auditing: a guide for DBA's, information security administrators and auditors, Elsevier Digital Press, 2009 (Available online through UOW Library)