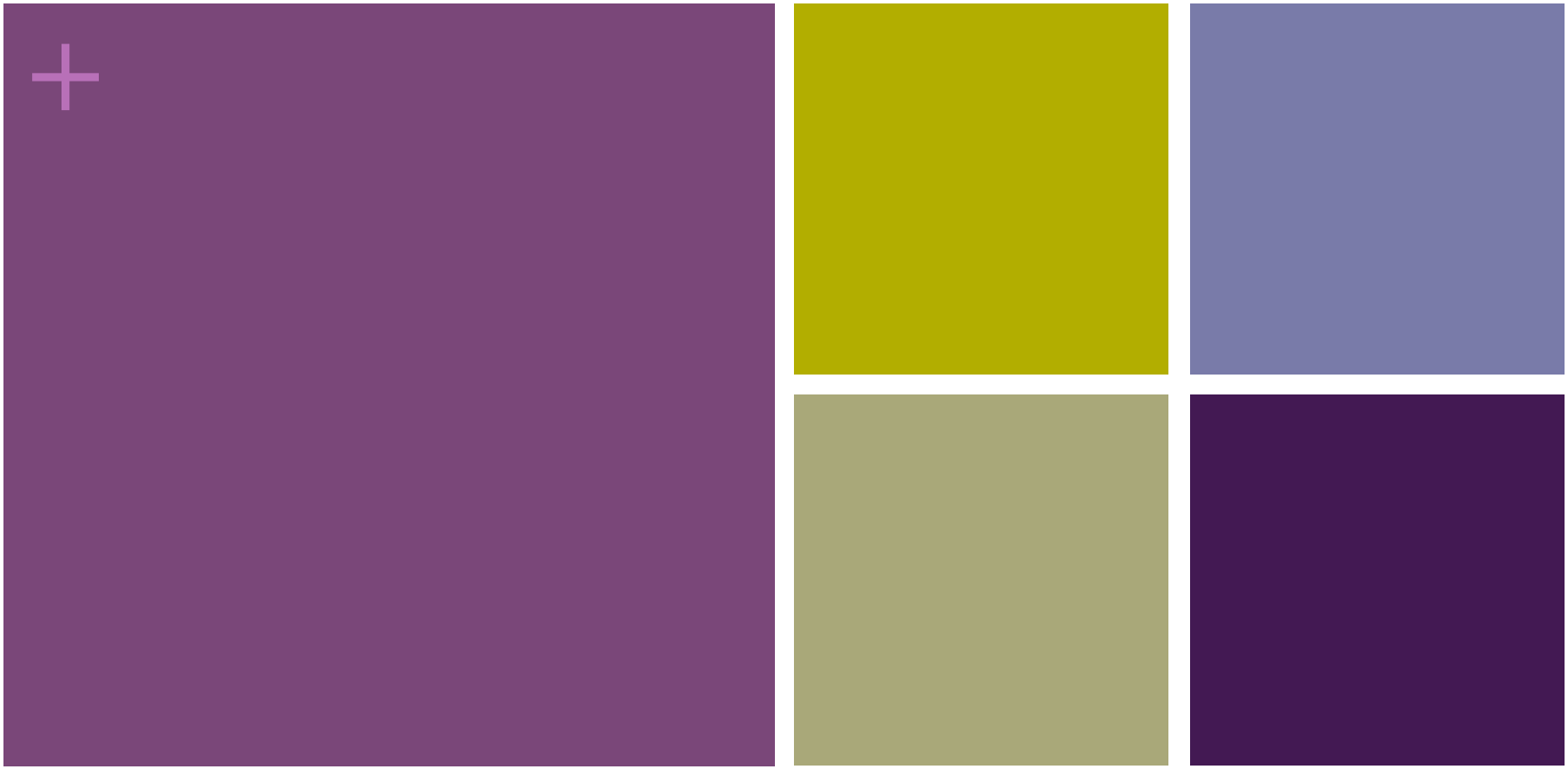


# Database Security

*Chapter 7*
*SQL Injection I: Identification*

# Objectives

- Describe an SQL injection and identify how injections are executed
- Identify how a Web application works and its role in SQL injections
- Define how to locate SQL vulnerabilities using error messages
- Apply inferential testing
- Review source code manually to locate injection vulnerabilities

# Objectives (cont'd.)

- Describe methods for automatic traversing of source code to locate injection vulnerability

# Understanding SQL Injections

- SQL injection
  - Method used by intruders to break into databases and Web sites
  - Intruders use bits of SQL code and SQL queries to gain database access
  - Creates vulnerabilities and can allow intruder to obtain full administrator privileges
- Three common strategies for SQL injections
  - Single channel
  - Multichannel
  - Observational

# Understanding SQL Injections (cont'd.)

- Single channel attack
  - Intruder uses one channel to execute SQL injections and obtain the returned results
  - Example: entering SQL injections into a Web application
- Multichannel attack
  - Intruder uses one avenue to initiate the injection
    - Uses a different channel to obtain results
- Inferential injections
  - Intruder does not intend to receive data
    - Observes and learns from returned behavior

# Injections and the Network Environment

- Most SQL injections performed through a Web application
  - Application interfaces with back-end database
- Web applications are common today
  - Examples: e-mail access, auctions, shopping, banking, blogging, online gaming
  - Primary target for intruders

# Injections and the Network Environment (cont'd.)

- General steps for retrieving and manipulating data using Web applications
  - User accesses the specific Web site
  - Site displays forms and fields for user input
  - Form resides on Web server and uses HTML and scripting language
  - Scripting language reacts to user's submission
  - SQL statements passed to company's application, or middleware server

# Injections and the Network Environment (cont'd.)

- General steps for retrieving and manipulating data using Web applications (cont'd.)
  - Middleware server acts as interface to the database
  - Database receives the query and returns results to application server
  - Application server returns results to the scripting language on the Web server
  - Scripting language, along with HTML, displays results on the screen

# Injections and the Network Environment (cont'd.)

- SQL injections deployed at beginning of process
- Two ways SQL injections cause destruction
  - Lethal SQL code placed into user input fields and executed at the database
  - Ill-written code sent to be stored in the database
- Primary method of detecting injections
  - Ensure application validates the user's input before sending it to the database
- SQL Server, MySQL, and Oracle
  - All at risk if applications are not properly secured

# Injections and the Network Environment (cont'd.)

- Dynamic SQL statement
  - SQL statement generated on the fly by an application using a string of characters from user input
  - Developers build applications that handle most of the SQL code in real time
- Static SQL statement
  - Statement built by the user in which full text is known at compilation

# Injections and the Network Environment (cont'd.)

- Example of differences between dynamic and static SQL statements
  - Web application displays input fields to users
  - Users fill in criteria and application uses these to search the database
  - Three fields: name, title, and department
  - Static SQL statement would need to take into account all possible combinations of user input
  - Dynamic SQL statements are best for Web database access
    - Vulnerable if input is not validated

# Injections and the Network Environment (cont'd.)

- Example of SQL injection attack
  - User inputs syntax `' or '1' = '1--`
  - Often returns first entry in a given table
  - If attacker places above syntax in username and password fields:
    - Web application creates SQL statement that checks username and password table and uses the first entry as attacker's credentials
    - Always at least one user in the database, so attacker is authenticated
    - If administrator account is listed first, attacker has obtained administrator privileges

# Identifying Vulnerabilities

- Primary step toward securing data
- Security actions fruitless without knowledge of system weaknesses
- Play the role of intruder to find vulnerabilities
- Several different types of attacks
- Different areas of network can be attacked
- Different methods of deploying injections

# Inferential Testing for Locating SQL Injections

- Look for clues in behaviors returned from the database in response to a controlled attack
- Administrator must input parameters from the client side of the database environment
  - Observe database behaviors
  - Document abnormal responses
- Administrator must be familiar with application and Web browser behavior during normal data retrieval

# Using HTTP

- All network communication based on same basic principles
  - Client sends request to obtain resource
  - Request is received and processed
  - Client's privilege is checked to determine allowable permissions
  - Once approved, requested resource packaged and sent from server to client
- Each step handled using standards or protocols

# Using HTTP (cont'd.)

- Examples of protocols
  - TCP defines rules to ensure reliable virtual connection
  - HTTP defines formatting method for hypertext requests and responses
- When request made from Web application to database server:
  - HTTP initiates TCP connection as transfer agent

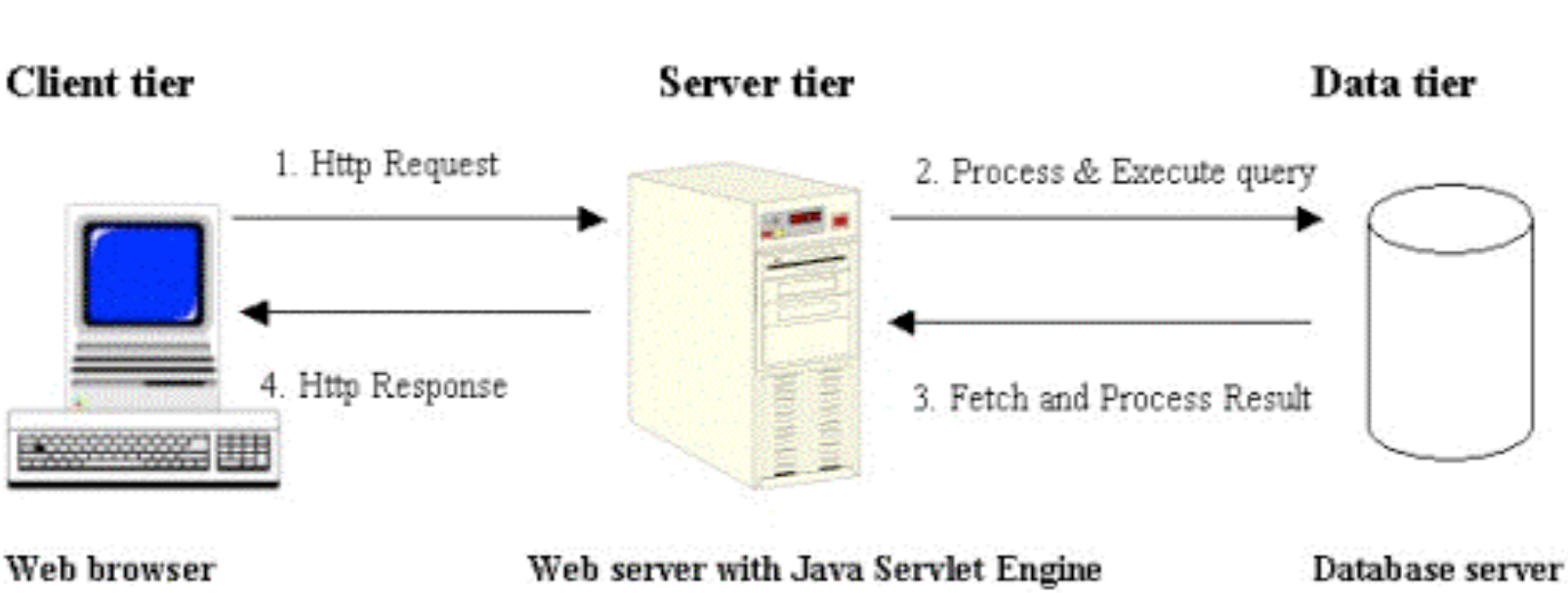
# Using HTTP (cont'd.)

- Eight predefined actions included in HTTP
  - HEAD, GET, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT
  - Two actions relevant to SQL injections: GET and POST
- GET requests
  - Encoded by browser into a URL
  - Server executes parameters appended to the URL itself

# Using HTTP (cont'd.)

- POST requests
  - User input included in the body of the request
    - Not the URL
  - Application server recognizes POST action and searches body of request for statements to send to database to execute
- User requests can be intercepted during transit from Web application to Web server
  - User data can be switched with SQL injections
- Unauthorized user can place SQL statements into form fields

# Using HTTP


Client tier
Server tier
Data tier
1. Http Request
2. Process & Execute query
4. Http Response
3. Fetch and Process Result
Web browser
Web server with Java Servlet Engine
Database server

# Using HTTP (cont'd.)

- Testing techniques
  - Third-party applications
    - Applications that intercept and manipulate HTTP data
  - Browser add-ons and plug-ins
    - Browser-specific applications that create specific interception
  - Proxy servers
    - Enables interception and modification of HTTP requests
  - Using one of the preceding tools to intercept and observe network's GET and POST parameters

# Determining Vulnerability Through Errors

- Malicious code not executed until it reaches the database
  - Nothing gets in the way of returning unauthorized results
- Administrators testing the environment may be presented with errors
  - Scripting language determines how to present the error
- Need strong familiarity with errors and how they are presented to find vulnerabilities

# Determining Vulnerability Through Errors (cont'd.)

- Ways to handle errors
  - Code application's scripting language to display specific error messages
    - These can give intruders information about the system
  - Configure Web applications to respond with generic messages
  - Choose not to handle errors at Web application
    - Allow HTTP to handle errors instead (Server Error)
- Using inference, professionals can note error handling location and message content
  - Use information to minimize vulnerabilities

# Typical Conditions with No Error

- First understand the typical error-free condition
- Example of typical or baseline configuration
  - Online grocer allows customers to search and purchase products on Web site *www.yum.com*
  - Customers choose categories and page displays list of available products
    - Dairy button is linked to URL: *http://www.yum.com/index.asp?category=dairy*
  - Scripting language receives output URL and sends statement to database requesting dairy items

# Typical Conditions with No Error (cont'd.)

- Example of typical or baseline configuration (cont'd.)
  - ASP request sent to database

```
food_cat = request("category")
sqlstr= "SELECT * FROM products WHERE
Food_Category = '"&food_cat&"'"
set rs-conn.execute(sqlstr)
```

  - ASP creates the SQL statement to be executed by the database

```
SELECT * FROM products WHERE Food_Category =
'Dairy'
```

# Typical Conditions with Typical Error

- Errors can occur under typical conditions
  - Response to common user error
  - Often overlooked because not viewed as threats
- Assume administrator changes URL manually to: *http://www.yum.com/index.asp?category=Hungry*
- ASP creates SQL statement to be executed by the database

```
SELECT * FROM products WHERE Food_Category =
'Hungry'
```

# Typical Conditions with Typical Error (cont'd.)

- Errors will be generated if Hungry is not a category of food within the system
  - Database returns error that column does not exist in the products table
- Does not necessarily indicate SQL injection vulnerability
  - URLs can be changed manually if database information is not hidden

# Injection Conditions with No Error

- Successful injection
  - SQL injection executed without error returned from database
  - Often due to Web applications that do not filter user input:
    - Or URLs that do not hide database information
- When testing for vulnerabilities:
  - Successful injections need immediate attention

# Injection Conditions with No Error (cont'd.)

- Statements often used in SQL injections that always return true

```
' or '1' = '1
' or 'ab' = 'a' + 'b (SQL Server)
' or 'ab' = 'a' ' b (MySQL)
' or 'ab' = 'a' \\' b (Oracle)
```

  - Statement that always returns false

```
' or '1' = '2
```

# Injection Conditions with No Error (cont'd.)

- Blind injections
  - Attacks made with little to no knowledge of the system
  - Series of true and false SQL statements used to attempt to discover information about a system
  - Difficult to detect due to their subtle nature
- Assume Yum URL can be manually changed to

*http://www.yum.com/index.asp?category=dairy 'or '1' = '1--*

# Injection Conditions with No Error (cont'd.)

- Changed URL results in SQL statement

```
SELECT * FROM products WHERE Food_Category =
'Dairy' 'or '1'='1 --
```

  - Database returns everything from the products table
  - Single quote at beginning of statement is correct syntax to complete a statement
  - Double dashes at the end starts a comment in SQL
    - All values that follow are ignored by the system
- Single quote characters can be inserted in different places to determine vulnerability

# Injection Conditions with No Error (cont'd.)

- If database reacts in same way with single quotes added to the SQL statement as it reacts without changes to the statement
  - Indicates vulnerability exists
- Syntax rules different between SQL languages

# Injection Conditions with Injection-Caused Error

- Some error messages clearly indicate that a vulnerability exists
  - Messages differ between database types
  - Also depends on how environment's application is built to handle errors
- Example of using incorrect syntax to cause an error
  - Assume Yum URL is manually changed to *http://www.yum.com/index.asp?category=dairy "or "1" = "1 --*

```
SELECT * FROM products WHERE Food_Category =
'Dairy' "or "1"="1 --
```

# Injection Conditions with Injection-Caused Error (cont'd.)

- Error would be returned
  - Syntax error due to double quotes appended to the SQL statement from the SQL injection
  - Often a result of blind injections
    - Attacker attempting to determine the dialect of SQL, type of database, and error handling of the Web application

# Generic Error Messages

- Generic error messages often have no reference to type of event that returned the error
  - Or no message will be presented at all
- Poses a challenge to intruders and database administrators
  - Difficult to determine if error has occurred in the database or the application
- To rule out an error within application error-handling system:
  - Administrator uses SQL statement that tests the database error returns

# Direct Testing

- Attacker's next goal
  - Test theories made during inferential testing
  - Actively execute SQL injections
- Active testing
  - Determines how far an intruder is able to reach into the system
  - Determines to what extent unauthorized access can be obtained
  - Determines how much data is available for view
  - Prepares administrator for removal of the injection

# Using the Code for Locating SQL Injections

- Source code analysis
  - Second most common approach to locating SQL injection
  - Requires less time and fewer resources than inferential or direct testing
- Administrator works with application developer
  - Ensures dynamic statements are being created and filtered without SQL vulnerability
  - Determines how and where user input is being accepted

# Source Code Analysis

- Analyzing source code
  - Can be a tedious and painstaking task
    - Can take months
  - One of the most effective ways to manage SQL injection vulnerabilities
- Tools are available to automate the process
  - Focus primarily on security
  - Weak in locating errors that SQL injections can exploit

# Source Code Analysis (cont'd.)

- Dynamic analysis
  - Attempt to find errors or vulnerabilities in source code while it is being executed
- Static analysis
  - Effort to find problems while program is inactive
  - Requires less expense and fewer resources
  - More effective at identifying vulnerabilities
- Problem areas
  - Poorly written functions
  - Unverified user input

# Source Code Analysis (cont'd.)

- Effective technique for reducing SQL injection attacks
  - Add filtering processes between user input and dynamically created statement
- Follow path of variable back to its origin
  - Place where user input enters the code
  - Determine whether restrictions have been placed on user input

# Source Code Analysis (cont'd.)

- Example
  - User inputs information into a text field of a Web application
    - Text field name is TFName
  - Variable in scripting language is defined, named, and assigned to TFName (UserInput = "TFName")
  - If UserInput is not verified to have appropriate set of characters, anything can be inserted into the text field and transferred directly to SQL statement

# Source Code Analysis (cont'd.)

- Consider how data is transferred from form into scripting language
  - Transfer involves HTTP actions GET and POST
  - Review HTML code to ensure desired settings
  - Research predefined functions specific to scripting language to ensure that validation is being applied

# Tools for Searching Source Code

- Tools are available to help facilitate source code review
  - None are as effective and reliable as manual searching
- Three methods for analyzing static source code
  - String-based pattern matching
  - Lexical token matching
  - Data flow analysis

# String-Based Matching

- Simple detecting tool
- Searches for and locates user-defined strings and patterns within source code
- Most basic of the three strategies
  - Produces highest number of false results
- Signatures are created for typical SQL injection variables
  - String-based matching systems attempt to find strings that match these signatures

# Data Flow Analysis

- Method for obtaining information about the way variables are used and defined in a program
- Determines the dynamic behavior of a program by examining its static code
- Source code divided into blocks of data
- Data flow analysis uses control flow graphs to display how events are sequenced
- Data flow analyzers map each variable and assigned value at each step in the program

# Lexical Analysis

- Lexical scanning
  - Process by which source code is read from left to right
  - Source code is grouped into tokens, based on some type of similar criteria
- Lexical analyzer can identify common symbols defined by initiating programming language

# Summary

- SQL injections can exploit vulnerability and allow intruders to obtain full administration of a database
- Two types of SQL statements include static and dynamic
- Administrators can test for SQL injections by sending a series of requests and observing the server reaction
  - Or by reviewing source code
- HTTP and TCP are protocols for sending information over the network

# Summary (cont'd.)

- HTTP data can be intercepted and manipulated
- Applications and databases handle errors differently
  - Error handling responses can give clues to intruders
- Source code can be analyzed while the program is static, or dynamically as the program is being executed
- Automatic source code analysis should be used by security professionals in conjunction with manual source code reviews