Chapter 7 SQL Injection Exploitation and Defense

**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
* Describe methods for automatic traversing of source code to locate injection vulnerabilities (易受攻击)

**Understanding SQL Injections**

SQL injection

* Method used by intruders to break into database 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

1. Intruder uses on a channel to execute SQL injections and obtain the returned results
2. Example: entering SQL injections into a Web application

* Multichannel

1. Intruder uses one avenue to initiate the injection
2. Uses a different channel to obtain results

* Observational (inferential injections)

1. Intruder does not intend to receive data
2. Observes and learns from returned behavior

**Injections and the network environment**

* Most SQL injections performed through a Web application

1. Application interfaces with back-end database

* Web application are common today

1. Examples: e-mail access, auctions, shopping, banking blogging, online gaming
2. Primary target for intruders

* General steps for retrieving and manipulating data using web applications

1. User accesses the specific web site
2. Site displays forms and fields for user input
3. Form resides on Web server and uses HTML and scripting language
4. Scripting language reacts to user’s submission
5. SQL statements passed to company’s application, or middleware server
6. Middleware server acts as interface to the database
7. Database receives the query and returns results to application server
8. Application server returns results to the scripting language on the Web server
9. Scripting language, along with HTML, displays results on the screen

* SQL injections deployed at beginning of process
* Two ways SQL injections cause destruction

1. Lethal SQL code placed into user input fields and executed at the database
2. iii–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

* Dynamic SQL statement (有user input)

1. SQL statement generated on the fly by an application using a string of characters from user input
2. 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

* Example of differences between dynamic and static SQL statements

1. Web application displays input fields to users
2. Users fill in criteria and application uses these to search the database
3. Three fields: name, title, and department
4. Static SQL statement would need to take into account all possible combination of user input
5. Dynamic SQL statements are best for web database access

Vulnerable if input is not validated

* Example of SQL injection attack

1. User inputs syntax ‘ or ‘1’ = ‘1--
2. Often returns first entry in a given table
3. If attacks places above syntax in username and password fields:
4. Web applications creates SQL statement that checks username and password table and uses the first entry as attacker’s credentials
5. Always at least one user in the database, so attacker is authenticated
6. 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 weakness
* 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

1. Observe database behaviors
2. Document abnormal response

* Administrator must be familiar with application and web browser behavior during normal data retrieval

**Using HTTP**

* All network communication based on same basic principles

1. Clients sends request to obtain resource
2. Request is received and processed
3. Client’s privilege is checked to determine allowable permissions
4. Once approved, requested resource packaged and sent from server to client

* Each step handled using standards or protocols
* Examples of protocols

1. TCP defines rules to ensure reliable virtual connection
2. HTTP defines formatting method for hypertext requests and responses

* When request made from Web application to database server:

HTTP initiates TCP connections as transfer agent

* Eight predefined actions included in HTTP

1. HEAD, GET, POST, DELETE, TRACE, OPTIONS, CONNECT
2. Two actions relevant to SQL injections: GET and POST

* GET requests

1. Encoded by browser into a URL
2. Server executes parameters appended to the URL itself

* POST requests

1. User input included in the body of the request

Not the URL (value is hidden)

1. 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
* Testing techniques

1. Third-party applications

Applications that intercept and manipulate HTTP data

1. Browser add-ons and plug-ins

Browser-specific applications that create specific interception

1. Proxy servers

Enables interception and modification of HTTP requests

1. 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
* ways to handle errors

1. code application’s scripting language to display specific error messages

These can give intruders information about the system

1. configure web applications to respond with generic message
2. 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

Web error message 会告诉骇客有用的信息. 因此, 不应该出现这些error message.

**Typical conditions with no errors**

* First understand the typical error-free condition
* Example of typical or baseline configuration

1. Online grocer allows customers to search and purchase products on web site [www.yum.com](http://www.yum.com)
2. Customers choose categories and pages displays list of available products

Dairy button is linked to URL: <http://www.yum.com/index.asp?category=dairy>

1. Scripting language receives output URL and sends statement to database requesting dairy items
2. ASP request sent to database

*Food\_cat = request(“category”)*

*Sqlstr = “SELECT \* FROM products WHERE*

*Food\_Category = ‘ “&food\_cat&” ’ ”*

*Set rs-conn.execute(sqlstr)*

1. ASP creates the SQL statement to be excited by the database

*SELECT \* FROM products WHERE Food\_Category = ‘Dairy’*

**Typical conditions with typical error**

1. Errors can occur under typical conditions

* Response to common user error
* Often overlooked because not viewed as threats

1. Assume administrator changes URL manually to: <http://www.yum.com/index/asp?category=Hungry>
2. ASP create SQL statement to be executed by the database

*SELECT \* FROM products WHERE Food\_Category = ‘Hungry’*

1. Errors will be generated if Hungry is not a category of food within the system

Database returns errors that column does not exist in the products table

1. Does not necessarily indicate SQL injection vulnerability

URLs can be changed manually if database information is not hidden

**Injection conditions with no errors**

* 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

* 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*

* Blind injections

1. Attacks made with little to no knowledge of the system
2. Series of true and false SQL statements used to attempt to discover information about a system
3. 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--
* 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 statements 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
* If database reacts in same way with single quotes added to the SQL statement as it reacts without changes to the statement

Indicates vulnerability exits

* Syntax rules different between SQL languages

**Injection conditions with injection-caused error**

**(根据这些错误, 骇客可以判断出这是什么数据库)**

* Some error messages clearly indicate that a vulnerability exits
* 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--

* 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

1. Test theories made during inferential testing
2. Actively execute SQL injections

* Active testing
  1. Determines how far an intruder is able to reach into the system
  2. Determines to what extent unauthorized access can be obtained
  3. Determines how much data is available for view
  4. 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 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

* 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
* Problems areas
* Poorly written functions
* Unverified user input
* Effective technique for reducing SQL injections 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
* Example
* User inputs information into a text field or a web application
* Text field name is TFName
* Variable in scripting language is defined, named, and assigned to TFNmae (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
* 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 analysis may 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
* HTTP data can be intercepted and manipulated
* Applications and database 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