INTRO TO ETHICAL HACKING

MIS 5211.701

Week 11

Site:

http://community.mis.temple.edu/mis5211sec701fall2018/

Tonight

- SQL Injection
- More Ruby
  - Permissions
  - Bounds checking

SQL Injection

- We are going to cover some "Basics"
- SQL Injection is a subset of the general flaw "Injection" covered last week
- Client supplied data passed to an application without appropriate data validation
- Processed as commands by the database
- Remember in all of this that we can also use the intercepting proxy to "add" text the browser doesn't want to accept

Frequently Used To:

- Perform operations on the database
- Bypass authentication mechanisms
- Read otherwise unavailable information from the database
- Write information such as new user accounts to the database

Caution

- Do not use your powers for evil.
- Ultimately, the reason for covering these attacks is to teach you how to prevent them.
- Well established sites are generally hardened to this type of attack.
- You might cause irreparable harm to a small "mom-and-pop" business.
- Even if you don't, breaking into someone else's database is illegal and unethical.

Brief SQL Review

- Querying tables:

  select column1, column2 **from** table_name;

  or

  select * **from** table_name;

- Conditions:

  select columns **from** table_name **where** condition;

Brief SQL Review

- Inserting new rows:

insert into table_name **values** (value1, value2);
or
insert into table_name **set** column1=value1, column2=value2, ...;

- Updating rows:

update table_name **set** column1=value1 **where** condition;

Brief SQL Review

- Deleting rows:

delete from table_name **where** condition;

- Set values in conditions:

select * **from** table_name
where column **in** (select_statement);

or

select * **from** table_name
where column **in** (value1, value2, ...);

Brief SQL Review

- Joining tables:

select * **from** table1, table2 **where** table1.attribute1 = table2.attribute2;

- Built-in Functions

select count(*) **from** test;

Brief SQL Review

- Pattern Matching
  select * **from** test **where** a **like** '%c_t%';
- Other Keywords
  select * **from** test **where** a **is null**;
- Metadata Tables
  - Highly vendor-specific
  - Available tables, table structures are usually stored in some reserved table name(s).

Form Specific to Version

- Different Vendor's Databases use different forms
- May want to use recon techniques to determine which database is in use
- What follows are some general techniques

Finding SQL Injection Bugs

- Submit a single quote ('), this is used in SQL as a string terminator and, if not filtered by the application, would lead to an incorrect query
- Submit a semicolon (;) this is used to end a SQL statement and, if it is not filtered, it is also likely to generate an error
- In either case:
  - If an error results, app is vulnerable.
  - If no error, check for any output changes.

Finding SQL Injection Bugs

- Can also try
  - Submit two single quotes (‘’).
    - Databases use ‘’ to represent literal ’
    - If error disappears, app is vulnerable
  - Comment deliminators (-- or /* */, etc)
  - SQL keywords like ‘AND’ and ‘OR’
  - String where a number is expected
    - Might also slip by SQL Injection detection system

Simple Example

- Assume actual SQL is
  - SELECT * FROM Users WHERE Username='$username' AND Password='$password’
- Now consider
  - $username = 1' or '1' = '1
  - $password = 1' or '1' = '1
- Becomes
  - SELECT * FROM Users WHERE Username='1' OR '1' = '1' AND Password='1' OR '1' = '1'

https://www.owasp.org/index.php/Testing_for_SQL_Injection_(OTG-INPVAL-005)

Simple Example (2)

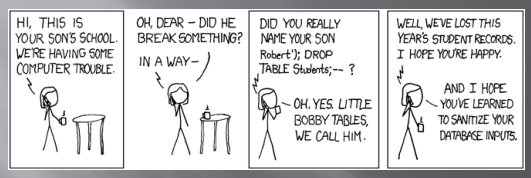
- Assume actual SQL is
  - SELECT * FROM products WHERE id_product=$id_product

    Or
  - http://www.example.com/product.php?id=10
- Now consider:
  - http://www.example.com/product.php?id=10 AND 1=2
- If you get a response that there are no matches try:
  - http://www.example.com/product.php?id=10 AND 1=1

Fingerprinting Databases

- Look at your error messages
- MySQL
  - You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '\'' at line 1
- SQL Server
  - ORA-00933: SQL command not properly ended
- PostgresSQL
  - Query failed: ERROR: syntax error at or near "'" at character 56 in /www/site/test.php on line 121.

Famous SQL Humor



http://xkcd.com/327/

Famous SQL Humor

http://gizmodo.com/5498412/sql-injection-license-plate-hopes-to-foil-euro-traffic-cameras

Permissions and Ruby

- Up until now we have sued a web portal to work through our Ruby scripts
- Now we we look at:
  - Permissions
  - Bounds checking
  - Input validation
  - Type checking
  - Parameter validation

Permissions

- Suppose we have a simple ruby script

puts "Hello!"

- Lets name in greeter.rb
- To run we would type ruby greeter.rb

- Lets say we just want to type greeter on the command line
- We would add #!/usr/bin/env ruby to the top of the file

Permissions

- #!/usr/bin/env ruby tells bash to
- Tells Bash what program to run our file with by asking for the current configured version of Ruby as specified by the env command
- For more on env, type "man env"

Permissions

- Now we need to tell the system this is an executable file
- Type "ls -l greeter.rb" at prompt
- Should see something like:
  - -rw-r--r-- 1 username staff 13 Feb 16 21:10 greeter.rb

Permissions

- Now typt "chmod 755 greeter.rb" at prompt
- Type "ls -l greeter.rb" at prompt
- Should see something like:
  - -rwxr-xr-x 1 username staff 13 Feb 16 21:20 greeter.rb
- The presence of x indicates that the file can be run directly without calling Ruby first
- The following command should get our file to say "hello."
  - ./greeter.rb

Permisions

- Almost there. Now, we just need to get rid of the prefix ./, which tells Bash where to look for greeter.rb
- Let's rename our file to just greeter
  - mv greeter.rb greeter

Permissions

- Next, at prompt type "echo $PATH" and look at the output
- Here's mine:
- /usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Applications/VMware Fusion.app/Contents/Public:/usr/local/share/dotnet:~/.dotnet/tools:/Library/Frameworks/Mono.framework/Versions/Current/Commands:/Applications/Xamarin Workbooks.app/Contents/SharedSupport/path-bin

Permissions

- Traditionally, any kind of user additions should be placed in /usr/local/bin/.
- If that folder doesn't exist, create it with:
  - mkdir -p /usr/local/bin/
- Last, either move the file to this directory or create a link or alias (Mac term)
- To create link:
  - ln -s $PWD/greeter /usr/local/bin/

Bounds Checking

- Before using variables, check to make sure the input is within scope
- I.e.
  - Is the string a string
  - Is the number a number, and the correct range
  - Etc..
- Includes
  - Type Checking
  - Input Validation
  - Parameter Validation

Next Week

- Web Services

Questions

?