CECS 378 Semester Project: SQL Injections

Christine Duong

Omar Kermiche

Alexander Ramirez

Justin Reid

John So

California State University, Long Beach

1. Omar Kermiche
2. Lead Coder - facilitated the creation of the SQL injection demo/project deliverables
   1. In order to start injecting SQL statements we needed a website. For the purposes of this lab and this project, this website had to be built specifically so we can test it/exploit it for site’s vulnerabilities (i.e. attacking it by injecting SQL). The only way that I’ve found that you can tinker with a website is if you make it yourself using the PHP/HTML coding languages.
   2. These are the few things that we needed before we could begin the implementation:
      1. Learn about PHP/HTML coding languages and syntax
         1. Previous to this project the whole team, myself included, had no idea how to code in PHP or HTML so we employed the help of various YouTube tutorials and coding language websites like geeksforgeeks.org. Ultimately what I learned is the PHP/HTML files that get created are the pages of a website
      2. How to create a server on my computer to store my PHP index files without spending a ludicrous amount of money
         1. The method that I found was using this application called **xampp** which is basically an app that allows you to use create and use a free local server on your computer by installing their built in feature called: **phpmyadmin** on your local windows explorer host and running apache and MySQL modules on that server. Phpmyadmin is basically an app that acts as a server on your localhost. It is also where you can create, store, and modify SQL databases
            1. For this project, I created a user table in phpmyadmin that has 5 columns: username, password, first name, last name, & phone number
         2. From there you create a new folder in xampp. This is the folder where all the pages of the website (php files) will get stored. You do this by finding the xampp folder location in your file explorer and clicking on the folder called **htdocs** then creating this new folder in there. Once the php index files are in the **htdocs** folder, xampp will link those php files to your local server
      3. A software IDE - or rather a PHP workspace that has more built in functionality that tells you if you made a syntax error or if there are any problems with your code
         1. First PHP IDE we tried using had limited functionality that didn’t show where your code bug was located so debugging was very challenging.
         2. I ended up finding and downloading an Eclipse IDE for PHP developers and that solved that problem.
      4. The relationship between a PHP IDE and the server that I was using

4a. Another problem that I faced was the fact that I didn’t

know how my local server was related to the Eclipse IDE that I used to code the PHP files. I didn’t know whether I had to link my server to Eclipse through eclipse itself.

* + - 1. It ended up being that I used the eclipse IDE to actually code the files and make sure there weren’t any bugs. Then I would manually add the php files 1 by 1 to the website folder I created in **2b.** in C://xampp/htdocs/<folder name>.
      2. Then I would go to my website by entering the address in my url: localhost/<folder name>. And this website would basically act as the console for which I can check the output for the code I wrote.

II. Alexander Ramirez

* 1. Role: PHP/MySQL developer
  2. Created an online MySQL server and an EC2 instance on AWS
     1. Used EC2 instance to run Apache
     2. Used FileZilla to transfer php files that displayed Log In page to EC2 instance.
     3. PHP connects to a MySQL database instance from the RDS server and executes statements
  3. How I connected to the EC2 instance
     1. I installed WinSCP and connected to the instance
     2. I used a ppk file provided by AWS and by inputted the server details as authentication. The host address was: ec2-3-15-191-104.us-east-2.compute.amazonaws.com  
        And the username was ‘ubuntu’
     3. A terminal to the server would open and I type the command: ‘sudo apt-get install phpmyadmin’ to install phpmyadmin onto the server
     4. I navigated to: ‘/var/www/html/’ to place that index.php file that will display when the site was visited
  4. Why using AWS didn’t work
     1. Using AWS service cost too much money.
     2. My billing statement for using a RDS and EC2 was in total $40.90. Maintaining the server was too expensive.
     3. Using FTP software and MySQLWorkBench had a high learning curve for my group mates.
     4. Hard to debug. We couldn’t test the code until we transferred the files to the server which was a hassle because the files had to be drag and dropped every time.
     5. With FTP, everybody had access to the master files. So if multiple people made different edits to the code, the server and code would ultimately have merge conflicts and unexpected errors/bugs.
  5. Setup local phpmyadmin environment
     1. I replicated Omar’s User table.
     2. I placed Omar’s index.php and script.php into ‘C:\xampp\htdocs\login’
     3. Now when I navigate to <http://localhost/phpmyadmin>, I will see the contents of index.php.
  6. Creating CSS file for index.php and script.php
     1. I created a style.css file that made the web pages more suitable to the eye. The CSS file modifies elements in both connect.php and index.php using div classes.
  7. Executing a simple SQL injection
     1. The two textboxes that take inputs will be used to inject unauthorized code.
     2. For the username textbox, anything typed in.
     3. In the password textbox, we input < ‘ OR 1=1; #> (Without the brackets)
     4. The original query being made was : <select \* from user where username = ‘[input from username text]’ and password = ‘[input from password text]’;>
     5. When we insert the sql injection, the query will be: <select \* from user where username=’’ and password=’’ OR 1=1;#’;>
     6. The injection continues after the opening single quote and closes it with another single quote.
     7. Another condition is added. The OR statement is followed by 1=1 which returns True. This means no matter what is inserted for username and password, the query will always return True.
     8. Since the condition is true, every row from user is returned since we selected all of them.
     9. OR 1=1 is followed by <;#>. The semicolon ends the statement and the ampersand comments anything followed by it. The last 2 characters from the original unaltered statement is then ignored.
  8. Union-Based SQL Injection
     1. In index.php, we utilize the textboxes once again. For the username, we type in nothing, but for the password, we type in: <’ UNION ALL select \* from user;#>
     2. The modified query will be <SELECT \* FROM user UNION ALL SELECT \* FROM user;#’;>
     3. The query will return all the rows from user combined with the rows from user whose username and password were blank.
     4. Normally entering nothing for username and password should return nothing but since we do another query of all the users using the UNION SQL operator, we still get the results.
  9. Error-based sql injection
     1. In index.php we utilize the textboxes. For the username, we type in nothing, but for the password, we type in: <’; SELECT extractvalue(rand(),concat(0x3a,(SELECT concat(password,0x3a,username) FROM user LIMIT 0,1))); #>
     2. The modified query will be <SELECT \* FROM user WHERE username = ‘’ and password = ‘’; SELECT extractvalue(rand(),concat(0x3a,(SELECT concat(password,0x3a,username) FROM user LIMIT 0,1)));#’;>
     3. The query will return nothing because it generates an error.
     4. The error message is used as an exploit to retrieve data.
     5. The error message returns where the syntax error occurs but the extractvalues return what data caused the error. We can select what data we want to generate the error.
     6. In this case, we used information about the creditcards table to return the error.
     7. Extractvalues expects to parse an XML string but since we concatenate “0x3a”, an error occurs. “0x3a” is the colon symbol in hexadecimal. Placing the colon in front of the query is invalid syntax.
  10. Credit Card table
      1. I created a new table in the sqlinjection schema called ‘creditcards’
      2. ‘Creditcards’ has the columns: nameOnCard, cardNumber, securityCode, expMonth, and expYear
      3. I created this new table to demonstrate the potential exploits that could done with a Union-based SQL injection

III. John So

1. Assumed the role of unofficial project manager. Created deliverables for project and facilitated communication and communication tools between group members. For communication we mainly used Discord, where we shared relevant videos and made progress of our project week by week.
2. Implemented an unofficial tracking schedule with the intent that the team as a whole stays up to date with the deliverables throughout the semester . Though we were far off our original schedule due to troubleshooting in creating our own server, we were eventually able to catch up to schedule last minute.
3. One of the main contributors and voices in the initial research/analysis phases of the project when the team was brainstorming on what cyber-attack/defense to implement for the project and how that cyber-attack/defense works. Originally, our group was not assigned to a specific topic, so we had the freedom to research and pick our own topic as long as it was not an attack/defense that was already assigned to the other groups. After brief research, our group came to the decision to make SQL injection our topic for the semester project.
   1. Some research that helped the lead coders/team members on grasping SQL injections:
      1. There are a wide variety of SQL injection vulnerabilities, attacks, and techniques, which arise in different situations. Some common SQL injection examples include:
         1. [Retrieving hidden data](https://portswigger.net/web-security/sql-injection#retrieving-hidden-data), where you can modify an SQL query to return additional results.
         2. [Subverting application logic](https://portswigger.net/web-security/sql-injection#subverting-application-logic), where you can change a query to interfere with the application's logic.
         3. [UNION attacks](https://portswigger.net/web-security/sql-injection/union-attacks), where you can retrieve data from different database tables.
         4. [Examining the database](https://portswigger.net/web-security/sql-injection/examining-the-database), where you can extract information about the version and structure of the database.
         5. [Blind SQL injection](https://portswigger.net/web-security/sql-injection/blind), where the results of a query you control are not returned in the application's responses.
         6. <https://portswigger.net/web-security/sql-injection#retrieving-hidden-data>
4. As the name implies, I needed to learn SQL in order to do this project. Conveniently, I am taking 323 Database Fundamentals this semester so I was able to learn SQL further down the semester.
   1. After reading a couple of articles and watching some videos on SQL injection, I tried to mimic what I observed on a server that Alex provided on Discord: [http://ec2-3-15-191-104.us-east-2.compute.amazonaws.com](http://ec2-3-15-191-104.us-east-2.compute.amazonaws.com/). However, I was unsuccessful in getting anything to work and also the server was not functioning.
5. For the powerpoint, I worked on the “Real World Example SQL Injection” and “Basics on How to Prevent SQL Injection.”
   1. Real World Examples
      1. Albert Gonzalez was truly an interesting case to research since he is accredited to having stolen one of the biggest number of debit and credit cards stolen. He and two unknown Russian hackers first did an SQL injection on HeartLand PaymentS Systems website and insert a packet-sniffing malware to obtain debit and credit card information.
      2. As for Anonymous hacking HBGary for freezing their donations to Wikileaks, Anonymous publicly humiliated HBGary and its CEO Aaron Barr. It first did SQL injection to obtain passwords, then reversed hashed them by rainbow table. They social engineered and eventually destroyed all their data.
   2. Preventing SQL Injection
      1. Besides string checking and automated software mentioned in the video, sanitizing inputs, parameterized statements, escaping inputs, Web Application Firewall(WAF), and avoiding dynamic SQL will cover the basic SQL injections.
         1. Sanitizing inputs is removing all potentially dangerous characters from input string including “, ‘, ?, =.
         2. Parameterized statements are that are ran during execution time, and it is crucial to secure them.
         3. Escaping outputs are properly escaping special string characters.
         4. WAF is an application firewall for HTTP applications and protects servers.
         5. Dynamic SQL are queries with user input. Since we assume that user input is dangerous, avoiding them at all cost will reduce the risk.

IV. Justin Reid & Christine Duong

1. Organized the presentation aspects including the video demo of the code running as well as the powerpoint setup.
   1. Decided to prepare a powerpoint slide to present first inform the audience about what actually is SQL Injection.
   2. Provide some history on some SQL Injection about when it was originated, and who were the first ones to attempt an attack
   3. List some examples of SQL Injection attacks done on major corporations or small business
   4. Present a demonstration of our SQL Injections
2. Decided on which SQL injections to use for the project
   1. Out of the many different types of SQL injections, we chose to do a Union-based, error-based, and classic injection.
      1. We chose to do a classic injection first because it was simple and we wanted to get a general sense of how the injections work.
      2. The union-based SQL injection was performed to show how harmful these injections can be. In our example, we collected users’ credit card information and this injection would expose all the details of them, which is dangerous if that information gets into the wrong hands.
      3. The error-based injection was chosen because we feel that it is an injection that is often overlooked by junior developers who might increase their site’s vulnerability due to leaving the SQL error messages displayed on the live site. Sometimes the injection is all an attacker needs to gain access to the entire database.
3. Struggling with creating a website meant for SQL injections
   1. As a group we struggled to get started with the injections because we needed a website that we could work with to test out our code. At first we tried to make our own, but we ran into a lot of problems such as the cost, functionality, and learning curve of it. That took up a few weeks and pushed us back in the timeline, but after sorting it out with a new website, we were able to complete our tasks and implement the 3 different types of SQL injections to present to the class.

Work Cited

<https://www.acunetix.com/websitesecurity/sql-injection/>

<https://www.acunetix.com/websitesecurity/sql-injection2/>

<https://dzone.com/articles/sqli-part-3-in-band-and-inferential-sqli>

<https://hydrasky.com/network-security/error-based-sql-injection-attack/>

<https://sqlwiki.netspi.com/injectionTypes/unionBased/#mysql>

<https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/#UnionInjections>

<https://www.hacksplaining.com/prevention/sql-injection>

<https://www.esecurityplanet.com/threats/how-to-prevent-sql-injection-attacks.html>

<https://www.csoonline.com/article/2124293/identity-theft-prevention-sql-injection-attacks-led-to-heartland-hannaford-breaches.html>

<http://www.cs.bu.edu/~goldbe/teaching/HW55812/jarib.pdf>

<https://www.youtube.com/watch?v=VaJSSVQUfaw&t=10s>