**Date:** 02/10/2018

**To:** Fengwei Zhang

**From:** Justin Sokana, Ian Wixcon, Patrick Veltri, Aaron Zhang, Paul Weliczko, Kristina Stevoff

**Subject:** Web Exploitation Project Proposal

**Problem Statement:**

****

SQL injection is used to exploit non-valid input vulnerabilities in databases. SQL injection is used to bypass logins on websites. The way it works is by typing in specific commands using specific characters such as ‘=’ and keywords such as ‘or’ in a login to bypass the login for websites without SQL injection preventatives. Right now many websites are very insecure and created by people with limited knowledge on the topic. An experienced hacker can easily destroy or delete an entire database with a single command. In 2012, 4 attack campaigns were performed each month for the average web application. Retailers received twice as many attacks as other industries. In 2017, SQL injection was the number one attack on the OWASP (Open Web Application Security Project) top 10 Application Security Risks.

Cross Site Scripting (XSS) is another exploit that one can use to harm websites. CSS and HTML are not programming languages, they are more relating to markdown than languages like Java or C. However, since the advent of JavaScript, web design has moved away from static pages and has actually implemented a language that is written along with the static CSS and HTML. A hacker that wishes to break into a website can simply input the text “<script> alert(“hello”) </script>” which will then change the browser itself. This means that even static pages are subject to exploitation.

**Related Work:**

****

Although nothing is a 100 percent guaranteed, SQL injection currently has 3 prevention rules. Rule 1 is to perform proper input validation. Rule 1 states to make sure the input is correct before proceeding. Rule 2 is to use a safe API (Application Programming Interface). Rule 2 recommends a parameterized interface. Rule 3 is for contextually escape user data. Rule 3 says do not allow special characters such as the equal sign.

HackerOne.com is a website that pays you money that finds flaws in websites that have an agreement with HackerOne.com. SQL injection is a flaw that could be addressed with HackerOne.com and you will get paid for it if the website if affiliated with HackerOne.com.

Web application firewalls that see suspicious inputs will block the input if the IP address has a bad reputational history. If the IP address has a repeated history of bad input, the web application firewall will block it. Other suggestions on how to reduce SQL injection exploits is to suppress error messages, regularly apply software patches, eliminate unnecessary database capabilities, and limited database privileges by context (securityplanet).

**New Solution:**

With SQL injection currently being the number one attack, we want to provide more insight on the topic of SQL injection so more programmers are aware of SQL injection. As a result, more programmers will be able to better protect themselves from SQL injection by putting in better safeguards.

We will explain the rules to prevent the exploitation. We will also show the difference between bad coding which allows the exploitation and good code which prevents it. XSS has the same easily avoidable mistakes as SQL and we plan to show how to avoid coding in those mistakes in the websites you make.

Our defense code will filter all user input, limit database privileges, and no external error statements (so exploiters have less knowledge of our code). Our bad code will not have these features. We then will show how each one of these features will protect our database from attackers.

**Evaluation Plan:**

Our plan is to make a mimic website using github.com as our collaborative platform. We will then use certain inputs with different variations of “or” and “=”. This will allow us to show how easily it is to perform SQL injection. We will also have a different variation of the code that is properly coded which prevents the SQL injection.

We also plan to show an XSS example on a static page to show that every website is vulnerable with the lack of security. We plan on going over the three rules as we explain our two versions of the code. With these rules, we will explain how they are implemented into the good code versus the vulnerable code.

**Sources:**

<https://www.owasp.org/index.php/Injection_Prevention_Cheat_Sheet>

<https://www.incapsula.com/web-application-security/sql-injection.html>

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

https://www.hackerone.com/