1. What is the paper about? What is/are the vulnerability? What causes the vulnerability?

The paper discusses SQL injection attacks, the techniques those wishing to exploit SQL injection vulnerabilities may use, and the existing detection/prevention techniques to guard against those attacks. SQL Injection attacks are considered one of the most serious threads for web applications as they may expose sensitive user information, which can lead to identify theft, fraud, and other security violations. Despite how serious the SQL Injection vulnerabilities are, they stem from a quite simple problem, insufficient validation/sanitation of user input. Developers have propose a wide range of techniques and coding guidelines to prevent these vulnerabilities, yet applications that are vulnerable to SQL injection attacks are still widespread.

1. What is/are the contributions of the paper? How was the vulnerability or insecurity discovered?

The paper discusses two main characteristics of any SQL Injection Attack: injection mechanism and attack intent. It uses these characteristics to not only give background information on SQL Injection Attacks, but to use when evaluating different attack techniques and evaluate known ways to defend/protect against the attacks.

First on the list is injection through user input. In this case, the attacker injects specially crafted user input, which will inject SQL commands. Injection can also occur through cookies. The cookie’s content can be used to build SQL queries and the cookie is under the client’s control, a malicious client could tamper with the cookie’s content to embed an attack in the cookie. Then if the cookie were used to restore the client’s state information, the attack would be submitted to the system. A third mechanism is injection through server variables. An attacker could create values for server variables placed in HTTP and network headers. The attack would be triggered if the server variables were logged to a database without sanitization. The paper also recognizes a final mechanism called second-order injection. In this case, the attackers would place malicious code/input into the system that will trigger a SQL injection attack when the user inputs certain data at a later time.

The first attack intent the paper discusses is identifying injectable parameters. The attacker tries to determine which parameters and user-input fields are vulnerable to an SQL injection attack. Attackers may also intend to perform database finger-printing, which will allow the attacker to know the version/type of database the application is using so database specific attacks can be crafted. The third possible intent of the attacker is to determine the database schema so the attacker can gain information such as table or column names, or column data types. Another attacker intent and the most common intent is to extract data values from the database which may include sensitive user information or data that is very useful for an attacker wishing to craft a larger scale attack. Other possible intents for an attacker include adding or modifying data in the database, executing remote commands, or evading detection. Attackers may also intend to shut down the database by denying service to others by locking or dropping database tables. Some other attackers wish to bypass authentication so they can assume the rights/privileges of another user who may have more access to sensitive data or even escalate the privileges of a user to gain access to sensitive data.

1. The detailed techniques to solve the problem.

Since the main cause of SQL injection vulnerabilities is insufficient input sanitation/validation, the most logical way to protect against attacks would be to eliminate the vulnerabilities with defensive coding. This would include input type checking, using functions that encode a string so as all meta-characters are specially coded to be interpreted by the database as normal characters, positive pattern matching where inputs are validated against known good input, and identification of all allowable input sources.

Through research, other techniques have been developed to help detect and prevent SQL injection attacks. Black Box Testing by Huang and colleagues uses a web crawler to identify all the points of vulnerability, creates attacks for each vulnerability, and then monitors how the application responds/reacts to the attacks. Static Code Checkers can be used to detect one of the main causes of vulnerabilities, improper type checking of input. Combined Static and Dynamic Analysis is used to not only analyze the code to determine different types of queries that an application can legally generate (static mode), but then takes those queries and uses them to compare incoming queries before they are sent to the database (dynamic mode). Taint Based Approaches analyze points in which preconditions have not been met and suggest ways to sanitize or filter for protection against possible attacks on these possible vulnerabilities. New Query Development Paradigms avoid SQL injection attack problems by changed the query-building process from a string concatenation, which can be unregulated, to a type-checked systemic process with rigorous user-input sanitization. Intrusion Detection Systems build models of all the typical queries and identifies runtime queries that do not match. Proxy Filters enforce input validation rules on data coming into the web application. Instruction Set Randomization creates queries using randomized instructions instead of normal SQL keywords, with a proxy filter to de-randomize the keywords before they are applied to the database.

1. What are the strength/weaknesses of the paper?

The first strength of the paper is the clarity in which they discuss the many different injection mechanism as well as the attack intents. I feel it is very important to know and understand both the way in which the attackers can exploit vulnerabilities, but also what they intended to gain from their attack. By looking at what the attackers were after that will help the developers put extra scrutiny on those items that may give attackers access to the information they are after and protect any avenues for them to obtain it.

Another strength is they assessed different tools for the prevention/protection not only for how well they did their intended job, but whether or not base code needed to be modified, whether detection and prevention were automated or not, and identified any additional infrastructure that might be necessary.

The first weakness I could see with the paper was they evaluated the techniques based on certain tools, but did not really discuss the effectiveness of the overall technique in general. One technique may be highly effective when used appropriately, but no tools may yet exist to provide an effective means to apply the technique.

The second weakness I would identify is they did not spend as much time talking about defensive coding practices even though use of these practices address the root cause of most SQL injection vulnerabilities and is the most straightforward solution.

1. What can you do better?

One improvement to TaintCheck would be to share any the signatures generated between programs. This would help other programs detect the attack and keep other vulnerable programs or systems from being infected in the first place.

In addition, I would include support for branch functions as they give the attacker a way to mask their program flow, which can allow malicious code to go undetected.

In this paper, we have presented a survey and comparison of current techniques for detecting and preventing SQLIAs. To perform this evaluation, we first identified the various types of SQLIAs known to date. We then evaluated the considered techniques in terms of their ability to detect and/or prevent such attacks. We also studied the different mechanisms through which SQLIAs can be introduced into an application and identified which techniques were able to handle which mechanisms. Lastly, we summarized the deployment requirements of each technique and evaluated to what extent its detection and prevention mechanisms could be fully automated.

Our evaluation found several general trends in the results. Many of the techniques have problems handling attacks that take advantage of poorly-coded stored procedures and cannot handle attacks that disguise themselves using alternate encodings. We also found a general distinction in prevention abilities based on the difference between prevention-focused and general detection and prevention techniques. Section 6.4 suggests that this difference could be explained by the fact that prevention-focused techniques try to incorporate

defensive coding best practices into their attack prevention mechanisms.

Future evaluation work should focus on evaluating the techniques’ precision and effectiveness in practice. Empirical evaluations such as those presented in related work (e.g., [17, 36]) would allow for comparing the performance of the different techniques when they are subjected to real-world attacks and legitimate inputs.