SCRAPEMAP: A CONTRIBUTION TO THE SQL INJECTION TOOL SQLMAP *

Richard Rich
Student
Texas A&M University-San Antonio
San Antonio, TX
email: richard.k.rich.ii@gmail.com

ABSTRACT

This report details the development and application of a novel extension tool for SQLMap, designed to enhance the detection and mitigation of SQL injection vulnerabilities. SQLMap, an established tool in the cybersecurity realm, primarily focuses on automating the process of detecting and exploiting SQL injection flaws. The new extension builds upon this functionality by introducing advanced data analysis capabilities, specifically tailored to identify and handle sensitive information exposed during SQL injections.

The extension employs sophisticated pattern recognition algorithms to efficiently scan and analyze extracted data for critical information, such as personal identifiers and financial details, thereby addressing significant security concerns in real-time. This enhanced functionality not only augments SQLMap's existing capabilities but also provides cybersecurity professionals with a robust toolset to perform more comprehensive security audits.

The broader application of this tool spans across various industries where data security is paramount, particularly in sectors handling large volumes of sensitive user data. By automating the detection of sensitive data leaks during SQL injection attacks, the extension significantly reduces the risk of data breaches, enhances compliance with data protection regulations, and strengthens overall cybersecurity postures.

This report underscores the extension's utility in real-world scenarios, demonstrating its effectiveness through a series of tests and case studies. By integrating this tool with SQLMap, users can achieve a more nuanced understanding of SQL injection threats and better safeguard their systems against one of the most prevalent security vulnerabilities in web applications today.

***K*eywords** SQLInjection · SQLMap · Data Security

1 Gap Identification

SQLMap is a powerful tool for database vulnerability analysis and data dumping but does not specialize in categorizing or filtering the dumped data for specific sensitive information types. My tool fills this gap by providing a focused analysis on identifying critical personal data, thereby extending SQLMap's utility in data security assessments.

2 Enhanced Utility / Implementation Highlights

By automatically parsing dumped data for sensitive information, my extension saves time and increases efficiency in security analysis workflows. It addresses a direct need for cybersecurity professionals to quickly identify potentially exposed sensitive information. My extension significantly broadens the utility of SQLMap by incorporating the ability to detect and analyze sensitive information such as email addresses, credit card numbers, and Social Security Numbers

* *Citation*: **Authors. Title. Pages.... DOI:000000/11111.**

(SSNs). I have defined specific regular expressions to accurately identify these types of data within scraped web content, or analyzed files. This capability is essential for conducting comprehensive security audits and ensuring data compliance across different platforms.

See Section 5.

3 Data Uses and Code Logic

In my extension, data is either fetched from websites asynchronously using `aiohttp` or read from local files, supporting both text and CSV formats. This approach leverages the efficiency of Python's asynchronous capabilities to enhance performance, particularly when scraping multiple websites concurrently.

3.1 Data Parsing and Analysis

The core of my tool revolves around the use of Python's `re` (regular expression) module to identify patterns within text data that correspond to common formats for emails, credit cards, and Social Security numbers (SSNs). The `PATTERNS` dictionary defines these regular expressions as follows:

- Emails: Matches standard email formats using characters, digits, underscores, plus signs, and hyphens before an "@" symbol, followed by a domain name.
- Credit Cards: Looks for sequences of four groups of four digits, which may be separated by spaces or dashes, resembling common credit card number formats.
- SSNs: Identifies U.S. Social Security numbers formatted as three digits, followed by a dash, two digits, another dash, and four digits.

The content fetched or read is then analyzed using these defined patterns. Matches are stored in a dictionary, making the data easy to report or further process. Each type of sensitive data has its own pattern, and matches are collected into a list under their respective categories in the `findings` dictionary, which is returned for reporting.

3.2 Data Transformation and Machine Learning

For website data, as well as data from files, the content is optionally transformed into a dataset using a `CountVectorizer`, which converts text data into a format suitable for machine learning analysis. This dataset is then used to train a Multinomial Naive Bayes model, adding a machine learning aspect to the tool. This model can be tailored for classification tasks involving text data, providing a robust framework for identifying and classifying sensitive information based on the learned patterns.

4 Integration With SQLMap

My tool is designed to complement SQLMap by analyzing the output files generated from SQLMap's data dumping process. Specifically, after running a SQLMap command to dump database contents into files, my tool can be used to analyze these files for sensitive information.

The integration with SQLMap is straightforward yet effective: my tool acts as a post-processing step to SQLMap's data dumping. It does not modify SQLMap's operations or interact with its processes directly; instead, it leverages the output files (`-dump`) that SQLMap generates. This approach allows for flexibility and ease of use, as it can be applied to any file generated by SQLMap without needing integration into SQLMap's core functionality.

Integrating this extension with SQLMap allows users to leverage SQLMap's powerful database penetration testing tools in conjunction with sensitive data analysis. This integration means that during a security audit, not only can SQL injection vulnerabilities be detected, but the security of the data itself can be assessed directly. This dual capability makes the tool more robust and comprehensive.

5 User Interface and Experience

The user interface for my tool is command-line based, ensuring simplicity and accessibility for users familiar with SQLMap and command-line operations. Users start the tool by entering the path to a dumped data file when prompted. This design choice keeps the usage straightforward and focused on the primary function of data analysis.

My tool enhances user interaction by providing clear, readable output that categorizes and lists found instances of emails, credit cards, and SSNs. By using a `set` to remove duplicate matches before printing, the tool also ensures that the output is concise and free of redundant information, making it easier for users to assess the sensitive data exposed in the dumped files.

By following this usage flow, users can easily deploy my tool in conjunction with SQLMap. For instance, after using SQLMap to dump the `users` table from a database, as shown in my provided command, they can then run the tool to analyze the dumped files for sensitive information, seamlessly integrating security analysis into their workflow.

This detailed explanation of my tool's implementation underscores its practicality and utility in enhancing data security analysis, especially in the context of assessing the exposure of sensitive information within large datasets dumped by SQLMap.

6 Impact and Evaluation

6.1 Efficiency Gains

My tool significantly reduces the time and effort required for security analysts to identify and mitigate potential data breaches. By integrating web scraping and machine learning, it can automatically search through databases and external web sources where sensitive data might be stored, displaying findings in a readable format. This automation eliminates the need for manual searches, streamlining the process of sensitive data identification.

6.2 Potential Use Cases

1. **Regulatory Compliance:** Organizations must comply with data protection regulations like GDPR, HIPAA, or CCPA. My tool enhances compliance checks by verifying the correct storage of sensitive information both within databases and scraped from relevant web sources. It can also identify data that should be encrypted or handled differently to meet compliance requirements.
2. **Data Breach Investigations:** In the aftermath of a data breach, quickly identifying exposed sensitive information is crucial. My tool can expedite this process by scanning both compromised databases and external web sources, pinpointing the specific data types accessed or stolen.
3. **Security Audits:** During security audits, the tool facilitates vulnerability identification by automating the search for sensitive data in databases and external sources, allowing auditors to focus more on securing data rather than locating it.
4. **Data Cleanup and Minimization:** My tool supports organizations in reducing data exposure risk by periodically scanning databases and web sources for sensitive data that is no longer needed or that should be deleted under data minimization principles.
5. **Mergers and Acquisitions:** During M&A processes, my tool aids in assessing the data practices and liabilities of the target company by identifying how and where sensitive information is stored, including data extracted from web sources, contributing to thorough risk assessments.
6. **Third-party Risk Management:** The tool can be used to audit vendors' databases as well as their public and private web interfaces to ensure they adhere to contractual and regulatory data protection standards.
7. **Development and Testing:** It assists developers and testers in secure software development environments, especially those using real data for testing, to ensure that sensitive data is adequately masked or sanitized, and not inadvertently exposed.

These use cases illustrate the broad applicability of my tool in enhancing data security, compliance, and operational efficiencies across different domains and scenarios. By integrating web scraping and machine learning, the tool not only aids in immediate security and compliance efforts but also supports a long-term strategy for data governance and risk management.

7 Future Enhancements

Looking ahead, I plan to expand the functionality of this extension in several ways:

- **Improve Pattern Accuracy:** Refine and expand the regex patterns to cover more types of sensitive information and reduce false positives. This involves adjusting current patterns and potentially adding new ones to better match the evolving formats of sensitive data.

- **Enhance Machine Learning Capabilities:** Improve the machine learning model by experimenting with different algorithms and feature extraction techniques to enhance its predictive accuracy and efficiency. This might include exploring more sophisticated models like Support Vector Machines (SVM) or deep learning approaches for text classification.
- **User Interface Enhancements:** Develop a more user-friendly interface, possibly integrating this extension directly into SQLMap's existing command-line interface or developing a standalone graphical user interface (GUI). This would make the tool more accessible to users who may not be comfortable with command-line operations.
- **Expand File Format Support:** Increase the types of file formats that can be analyzed, such as adding support for JSON, XML, and other commonly used data interchange formats. This would enhance the tool's versatility and make it capable of handling a broader range of data sources.
- **Robust Error Handling and Logging:** Implement more sophisticated error handling and logging mechanisms to better manage and debug issues during the scraping and analysis processes. Enhanced logging will help in maintaining a clear record of operations and errors, which is crucial for troubleshooting and improving the tool over time.

8 Comparison With jSQL and Havij

8.0.1 Tool 1: jSQL

Supported DBs

jSQL provides automatic injection of 33 database engines including: Access, Altibase, C-treeACE, CockroachDB, CUBRID, DB2, Derby, Exasol, Firebird, FrontBase, H2, Hana, HSQLDB, Informix, Ingres, InterSystems-IRIS, MaxDB, Mckoi, MemSQL, MimerSQL, MonetDB, MySQL, Neo4j, Netezza, NuoDB, Oracle, PostgreSQL, Presto, SQLite, SQL Server, Sybase, Teradata, and Vertica.

jSQL Functionality and Feature Set:

- **Multiple Injection Strategies:** jSQL offers various injection strategies including Normal, Stacked, Error, Blind, and Time. These strategies enhance jSQL's adaptability, allowing it to effectively identify vulnerabilities even in well-protected databases.
- **Parallel Bitwise Boolean Blind and Time Strategies:** These advanced strategies improve the efficiency and speed of Blind and Time-based SQL injections by conducting multiple checks in parallel, making the tool faster in confirming vulnerabilities.
- **Various Injection Processes:** jSQL offers different methods (or processes) for performing SQL injections, tailored to specific situations or to bypass certain types of security measures such as Default, Zip, and Dios.
- **Database Fingerprint and Script Sandboxes:** jSQL can identify the specific database being used (fingerprinting) and provides a safe environment (sandbox) for testing and refining SQL scripts and tampering techniques.
- **Multiple Target Injection and File Operations:** jSQL can target multiple databases or servers simultaneously and can read from and write files to the server, critical for assessing the impact of a vulnerability or for demonstrating data exfiltration.
- **Web Shell, SQL Shell, and Brute Force Hash:** jSQL can create web shells and SQL shells for interacting with the compromised server or database and includes a brute force feature for cracking password hashes.
- **Admin Page Search and Hash/Text Operations:** jSQL can search for admin pages and supports hashing, encoding, and decoding text, providing additional utility for analyzing and preparing data.
- **Authentication and Proxy Support:** jSQL supports various authentication mechanisms and proxy support (HTTP, SOCKS4, SOCKS5), ensuring users can operate in environments with different security configurations and route traffic through intermediaries.

This comprehensive feature set makes jSQL a powerful tool for SQL injection testing and vulnerability assessment.

8.0.2 Tool 2: Havij

Havij Supported DBs

Havij supports the databases MySQL, Oracle, PostgreSQL, Microsoft SQL Server, Microsoft Access, and Sybase.

Functionality and Feature Set

- **Complete HTTPS Support:** Havij fully supports HTTPS websites, enabling it to test and exploit SQL injection vulnerabilities on secure sites that use SSL encryption.
- **Regular Updates:** The tool receives various updates that introduce new features, enhance existing functionalities, and improve security measures to keep up with the evolving landscape of web security.
- **Added MS SQL Blind:** This feature enhances Havij's ability to exploit blind SQL injection vulnerabilities specifically in Microsoft SQL Server databases, allowing for more effective data extraction without visible errors.
- **Blind MSAccess (Commercial Version Only):** In the commercial version of Havij, there is support for exploiting blind SQL injection vulnerabilities in Microsoft Access databases, expanding the tool's versatility across different database systems.
- **PostgreSQL Support:** Havij includes support for testing and exploiting vulnerabilities in PostgreSQL databases, further broadening its applicability across various database platforms.
- **Easily Accessible User Manual:** The tool comes with a user manual that is easy to understand and follow, making it accessible to users of varying technical backgrounds.
- **Additional Dumping Data File Feature:** Havij allows users to dump database contents into files for offline analysis, enhancing the post-exploitation process by providing a way to analyze extracted data thoroughly.
- **XML Format for Data Storage:** The tool supports XML format for storing dumped data, offering a structured and widely compatible format for data analysis and reporting.
- **Option to Remove Logs:** Users have the ability to remove logs generated by Havij, helping maintain operational security and minimize digital footprints during testing.
- **Customizable Default Settings:** Havij allows users to change its default settings at any time, offering flexibility to tailor the tool's operation according to specific testing needs or preferences.
- **Repair Methods:** To address and mitigate identified vulnerabilities, Havij provides repair methods that can help secure websites against the types of SQL injection attacks that the tool can exploit.
- **Keyword Testing:** This feature enables more refined testing by allowing users to specify keywords for targeted testing, enhancing the precision and efficiency of vulnerability discovery.
- **Error Fixing Feature:** Havij includes functionalities to fix common errors encountered during SQL injection attacks, streamlining the exploitation process and reducing the need for manual troubleshooting.

Performance

Accuracy: Havij is known for its injection success rate, which is claimed to be more than 95% accurate at injection vulnerable targets, making it a powerful tool in the hands of penetration testers and malicious users.

Usability

Havij is easily accessible to new users. It is a point-and-click windows GUI application with an installer, which is a major advantage to inexperienced users.

Comparative Analysis

While both tools are powerful for SQL injection testing, jSQL Injection offers a broader range of database support and advanced technical features suited for a diverse set of scenarios and user proficiency levels. Its adaptability and comprehensive feature set make it a strong tool for in-depth security assessments.

Havij, on the other hand, emphasizes ease of use with its GUI and streamlined process for exploiting SQL injection vulnerabilities, making it particularly appealing for quick assessments or users with less technical background. Its focus on major databases and user-friendly design positions it as a practical tool for rapid vulnerability exploitation.

In terms of performance, both tools are designed to be efficient within their operational contexts, with Havij boasting a high success rate and jSQL offering speed and precision through its advanced strategies.

Usability is a key differentiator, with jSQL appealing to users who prefer a mix of GUI and command-line options and appreciate detailed configuration possibilities, while Havij caters to those seeking an accessible, point-and-click interface for immediate results.

8.1 Specificity in Identifying Sensitive Data

For my SQLMap extension, it is unique in its capability to parse and identify specific sensitive information (SSNs, credit cards, emails). Neither Havij nor jSQL Injection specifically advertises a feature solely dedicated to automatically identifying and reporting specific types of sensitive data like SSNs, email addresses, and credit card numbers directly within their standard toolset. These tools are primarily focused on discovering and exploiting SQL injection vulnerabilities, which can lead to unauthorized access to a wide range of data stored in a database, but they do not specifically filter or highlight sensitive information types by default.

9 Case Study Evaluation

9.0.1 Scenario 1: Database Vulnerability Assessment

Database Vulnerability Assessment

The controlled environment I used was the DVWA website to test these tools. The difficulty for SQL injection was set to low on this website, and the information provided for exploitation includes the pages GET request information such as the PHPSESSID and the GET requests URL, since the DVWA uses a user and password to log in, otherwise you will get a 302 redirect when attempting to perform SQL injection. To find the cookies, one must click on the SQL Injection page on the DVWA website and submit a request through the empty field on the page, then use their developer tools on the right-hand corner to use the information necessary for performing SQL injection on the page.

Data Extraction Efficiency

I used a controlled environment through the use of the DVWA website on my Kali Linux virtual machine to test how each tool extracts data. The performance measurement of jSQL injection did not find an injection vulnerability with the DVWA website when I ran it with the same cookie parameters (PHPSESSID, security level) as I did with the SQLMap tool. Havij was not able to run on the Kali Linux due to issues with compatibility to Linux applications. In efforts to run it on the Kali Linux Machine, Wine, a compatibility layer/emulator was installed, but the issues persisted.

Measured Data Extraction Performance for Tools

9.1 Real Time

Table real_time provides insights into the real-time performance of various tools in terms of the time taken (in seconds) to extract data. SQLMap, a widely-used tool, consumed 46.04 seconds for data extraction, indicating its relatively slower performance in real-time scenarios. In contrast, "My tool (ScrapeMap)" exhibited significantly better real-time performance, requiring only 8.23 seconds for the same task. Notably, Havij's performance is denoted as "n/a," indicating that data on its real-time performance was not available. jSQL Injection, another tool evaluated in this context, took 22.9 seconds for data extraction. This comparison sheds light on the varying efficiency of these tools in handling real-time data extraction tasks.

9.2 User Time

Table user time presents the user-time performance of various tools. In this context, user time refers to the time (in seconds) spent by the CPU executing user-level operations during the data extraction process. Among the evaluated tools, "My tool (ScrapeMap)" demonstrates the most efficient user-time performance, requiring only 2.76 seconds for data extraction. SQLMap follows with 8.01 seconds, indicating relatively slower user-time performance compared to "My tool." Notably, both Havij and jSQL Injection are denoted as "n/a," indicating that data on their user-time performance was not available. This comparison highlights the efficiency differences among the evaluated tools in terms of user-time performance.

9.3 System Time

Table system_time illustrates the system-time performance of various tools. System time refers to the time (in seconds) spent by the CPU executing kernel-level operations during the data extraction process. Among the evaluated tools, "My tool (ScrapeMap)" exhibits the most efficient system-time performance, requiring only 5.44 seconds for data extraction. SQLMap follows with 25.28 seconds, indicating relatively slower system-time performance compared to "My tool." However, both Havij and jSQL Injection are denoted as "n/a," signifying that data on their system-time performance was not available. This comparison provides insights into the efficiency differences among the evaluated tools in terms of system-time performance.

9.4 CPU Usage

Table cpu_usage provides insights into the CPU usage of various tools during the data extraction process. CPU usage is expressed as a percentage of the CPU's capacity that each tool utilizes. Among the evaluated tools, "My tool (analyze_dump)" exhibits the highest CPU usage, utilizing 99% of the CPU's capacity during the data extraction process. SQLMap follows with 72%, indicating a relatively lower CPU usage compared to "My tool." However, both Havij and jSQL Injection are denoted as "n/a," signifying that data on their CPU usage was not available. This comparison provides insights into the efficiency differences among the evaluated tools in terms of CPU utilization during data extraction.

Tool	Time (seconds)
SQLMap	46.04
ScrapeMap (My tool)	8.23
Havij	n/a
jSQL Injection	22.9

Table 1: Real Time Type

Tool	Time (seconds)
SQLMap	8.01
ScrapeMap (My tool)	2.76
Havij	n/a
jSQL Injection	n/a

Table 2: User Time Type

Tool	Time (seconds)
SQLMap	25.28
ScrapeMap (My tool)	5.44
Havij	n/a
jSQL Injection	n/a

Table 3: System Time Type

Tool	CPU Usage (%)
SQLMap	72%
ScrapeMap (My tool)	99%
Havij	n/a
jSQL Injection	n/a

Table 4: CPU Usage

10 Literature Review of "A Technical Review of SQL Injection Tools and Methods: A Case Study of SQLMap"

Summary

This paper explores the field of SQL injection vulnerabilities, offering an insight of various types alongside practical detection methods. It discusses the critical collection of tools available for identifying and preventing such pervasive attacks, emphasizing their significance in safeguarding against SQL injection breaches. Through a methodical comparison, this study evaluates these tools against key performance metrics, highlighting their effectiveness and operational aspects. Central to this investigation is the application of SQLMap, identified as an essential tool in this context, on a legally compliant and ethically sound testing environment. The deployment of SQLMap showcases its use in penetrating database defenses to successfully retrieve sensitive information, including usernames and passwords, highlighting the tool's utility in real-world cybersecurity defenses.

Structure of "Technical Review of SQL Injection Tools and Methods"

Structure

- Section two discusses the related work of the paper.
- Section three shows the SQL injection overview and detection approach.
- Section four discusses the SQL Injection Tools.
- Section five discusses the SQL injection Scenarios Study.
- Finally, Section six provides the conclusions and directions for future research.

Introduction

This literature review begins with a brief overview of SQL injection vulnerabilities and the importance of tools in exploiting these vulnerabilities. It sets the stage for discussing various tools used by attackers, highlighting the paper's focus on comparing these tools.

Tool Descriptions

Each tool is discussed in turn, detailing its functionalities, advantages, and limitations. The tools covered include Blind SQL Injection, Marathon, Havij, SQL Brute, Sqlninja, Absinthe, Pangolin, Sqlier, SQLsus, and SQLMap. For each tool, the description emphasizes how it works, its special features, and any unique limitations that restrict its use.

Comparative Analysis

Table of Comparison: Presents a comparative table (Table 4), summarizing the key features, database compatibility, and unique characteristics of each tool. This section is crucial for highlighting the differences and making it easy for readers to understand which tools excel in specific areas. The comparative criteria discuss important factors such as database type compatibility, automatic operation, machine learning support, and file system interaction as bases for comparison.

Application Case Study

This section details an actual application of one of them on a legal and ethical testing site, demonstrating how the tool can be used to access a database and retrieve sensitive information like usernames and passwords.

Discussion

This section discusses how the tools differ in their operation methods, including trial and error, cookie manipulation, password brute-forcing, and command execution on operating systems. Highlights the importance of certain features like machine learning and automatic operation in enhancing the tool's effectiveness.

Conclusion

The conclusion summarizes the key findings from the comparative analysis and case study. Emphasizes the criteria that are most important for attackers when choosing a tool, such as the ability to handle various database types and deal with system files.

10.1 Comparison With My Evaluation

Focus and Scope

While the my evaluation concentrates on enhancing SQLMap with a specific supplementary tool for detecting sensitive information, the second paper offers a broader analysis of many more SQL injection tools, comparing them on various criteria. It also discusses the implementation of machine learning that makes the automation of the SQL injection tools possible. The table in the first paper contrasted in the way it compared the different tools but shared the similarities in the databases that are supported. It focuses more on the requirements for automation that go into the tools, whereas my table describes the specific functionalities and features of the tools. The case study performed in the first paper did not focus on the comparison of the SQL Injection tools in terms of performance. My paper discusses the elapsed time for the tools and their overall performance in my case study.

In comparison to my paper, I discuss a tool I made that works with SQLMap to help find and show sensitive data like emails, credit card numbers, and social security numbers without needing to look through the whole database manually. I explained why I made this tool and how it fills a gap that SQLMap doesn't cover. I then discuss how my tool fits into the bigger picture with SQLMap, how easy it is to use, and when it could be really useful, like during safety checks or when looking into security breaches. After that, I compare my tool to others called jSQL and Havij, focusing on what databases they work with, their special features, how they find vulnerable spots, and how user-friendly they are. Finally, I put my tool, jSQL, and Havij side by side to see how they stack up against each other. I then presented a case study which I used the DVWA to test these tools and their performance of data extraction. I measured their performance of the tools that worked and put this data in a table format.

11 Literature Review of: "BlindCanSeeQL: Improved Blind SQL Injection For DB Schema Discovery Using A Predictive Dictionary From Web Scraped Word Based Lists"

The report "BlindCanSeeQL: Improved Blind SQL Injection For DB Schema Discovery Using A Predictive Dictionary From Web Scraped Word Based Lists" by Ryan Wheeler presents a thorough investigation into Blind SQL Injection, a technique utilized by attackers to extract data from databases through manipulation of SQL queries. Submitted as a thesis for the degree of Master of Science in Computer Science, the report delves into the complexities of SQL Injection Attacks (SQLIAs), with a particular focus on Inference, or Blind Injections.

Beginning with a dedication to John Draper, a prominent figure in technology, the report proceeds to explore the fundamentals of SQLIAs, defining them as code-injection attacks where malicious SQL statements are inserted into input fields for execution. It emphasizes the significance of addressing Blind Injections, a subset of SQLIAs, due to their potential severity.

While discussing countermeasures against SQLIAs, including parameterized statements, least privilege, stored procedures, and character encoding, the author argues for the insufficiency of these measures in isolation, prompting the need for advanced detection and mitigation techniques.

Introducing the BlindCanSeeQL (BCSQL) tool, developed in C# using Visual Studio 2012 and .NET Framework 4.5.1, the report showcases its functionality in targeting Boolean-based injections for MS SQL Server. Notably, BCSQL features a web crawler that constructs a dictionary file from HTML input tags, facilitating autocompletion during blind injection attempts.

Further elucidating the enhancements of BCSQL over existing tools, the report highlights key improvements such as sorting by length of object names, tracking current owner, modified bisection rules, and autocompletion based on the dictionary file. These refinements contribute to the tool's efficacy in schema discovery and injection detection.

Concluding with a comparative analysis between BCSQL and SQLMAP, another prevalent tool for detecting SQLIAs, the report substantiates BCSQL's superiority in terms of efficiency, particularly in reducing the number of requests made to the server for discovering object names within the database schema.

Conclusion

In summary, the report offers a comprehensive exploration of Blind SQL Injections and introduces BCSQL as a novel solution for enhancing detection capabilities. Through meticulous analysis and comparison with existing tools, the author demonstrates BCSQL's effectiveness in mitigating SQLIAs, thereby contributing significantly to the field of cybersecurity.

11.1 Comparison with My Article

The article discusses Blind SQL Injection, a technique used by attackers to extract data from a database by manipulating SQL queries. The article introduces a tool called BlindCanSeeQL (BCSQL) that improves the efficiency of detecting these attacks. BCSQL is developed in C# using Visual Studio 2012, .NET Framework 4.5.1 and targets Boolean based injections for MS SQL Server.

My article discusses a Python program that uses web scraping to extract data from websites. It uses the aiohttp and BeautifulSoup libraries to fetch and parse website content, and then uses regular expressions to search for specific patterns in the text, such as email addresses, credit card numbers, and social security numbers. It also includes functionality to train a machine learning model on the scraped data using the sklearn library. In summary, while both the article and the script in my article deal with data extraction, they do so in different contexts and with different methods.

The article focuses on a specific type of SQL Injection attack and how to detect it, while the script focuses on extracting data from websites through web scraping.

12 Acknowledgements

I extend my gratitude to Dr. Alsmadi for his contributions to the development of the SQLMap Extension tool. His suggestion to expand the primitive concept of the tool has been instrumental in enhancing its capabilities. This guidance has enabled the creation of datasets from testing websites, further facilitating their use in machine learning applications. Dr. Alsmadi's overall direction and insight have profoundly enriched the project, and for this, I am immensely thankful.

13 References

@articlebaklizi2022technical, title=A Technical Review of SQL Injection Tools and Methods: A Case Study of SQLMap, author=Baklizi, M. and Atoum, I. and Abdullah, N. and Al-Wesabi, O. A. and Otoom, A. A. and Hasan, M. A.-S., journal=International Journal of Intelligent Systems and Applications in Engineering, volume=10, number=3, pages=75–85, year=2022, url=https://www.ijisae.org/index.php/IJISAE/article/view/2141

@articlewheeler2015blindcanseeql, title=BlindCanSeeQL: Improved Blind SQL Injection For DB Schema Discovery Using A Predictive Dictionary From Web Scraped Word Based Lists, author=Wheeler, Ryan, year=2015