Vertti Huovila

Improving the Security of SQL Server using SQL-Map Tool

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

3 February 2024

Abstract

Author:	Vertti Huovila
Title:	Improving the Security of SQL Server using SQL-Map Tool
Number of Pages:	25 pages + 11 appendices
Date:	3 February 2024
Degree:	Bachelor of Engineering
Degree Programme:	Information Technology
Specialisation option:	IoT and Networks
Instructor(s):	Janne Salonen, Principal Lecturer

This thesis is about the improvement of SQL server security using SQL-MAP, an open-source penetration testing tool. The thesis discusses the importance of SQL server security, explains the complexity of SQL databases, their structure, and the concept of SQL injection. The use of TCP/IP, the NMAP tool and the SQL-MAP tool are discussed in detail. The objective is to demonstrate how the SQL-MAP tool can be used to improve SQL server security by detecting and mitigating SQL injection vulnerabilities. The result is a comprehensive guide detailing the use of SQL-MAP to improve SQL server security, supported by a practical demonstration of SQL-MAP with the help of Damn Vulnerable Web Application (DVWA) this is done so anyone that wants can follow the practical parts steps for themselves can try it out.

Keywords: SQL, SQL-MAP, SQL Injection

Tiivistelmä

Tekijä:	Vertti Huovila
Otsikko:	SQL-Palvelimen suojauksen parantaminen SQL-Map Työkalulla
Sivumäärä:	25 Sivua + 11 liitettä
Päivämäärä:	29.1.2024
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja Viestintätekniikka
Ammatillinen pääaine:	IoT ja tietoverkot
Ohjaajat:	Janne Salonen, Osaamisaluejohtaja

Tämä opinnäytetyö käsittelee SQL-palvelimen tietoturvan parantamista käyttämällä SQL-Mappia, joka on avoimen lähdekoodin penetraatiotestityökalu. Opinnäytetyössä käsitellään SQL-palvelimen tietoturvan tärkeyttä, selitetään SQL-tietokantojen monimutkaisuutta, niiden rakennetta ja SQL-injektion käsitettä. TCP/IP:n, NMAP-työkalun ja SQL-MAP työkalun käyttöä käsitellään yksityiskohtaisesti. Tavoitteena on osoittaa, miten SQL-MAP työkalun avulla voidaan parantaa SQL-palvelimen tietoturvaa havaitsemalla ja lieventämällä SQL-injektiohaavoittuvuuksia. Tuloksena on kattava opas, jossa kerrotaan yksityiskohtaisesti SQL-Mapin käytöstä SQL-palvelimen turvallisuuden parantamiseen, ja sen tukena on SQL-Mapin käytännön esittely Damn Vulnerable Web Application (DVWA) -ohjelman avulla. Tämä tehdään, jotta kaikki halukkaat voivat seurata käytännön vaiheita ja kokeilla niitä itse.

Avainsanat: SQL, SQL-MAP, SQL Injektio

List of Abbreviations

SQL: Structured query language

TCP: Transmission Control Protocol

IP: Internet Protocol

TLS: Transport Layer Security

MAC: Media Access Control

DVWA: Damn Vulnerable Web Application

VM: Virtual Machine

SQLi: SQL Injection

OS: Operating System

SSH: Secure Shell

DBMS: Database Management System

GDPR: General Data Protection Regulation

CI/CD Pipeline: Continuous Integration/Continuous Deployment

Cron Job: Scheduled task in Unix and Unix-like operating systems

1 Introduction

In a world revolving around information, organisations are vital in effectively managing and safeguarding information. As our reliance on databases grows, ensuring their security becomes a pressing concern. This involves mitigating numerous cyber threats, including the prevalent SQL injection vulnerability. In the pages ahead, the aim is to assist IT professionals, and anyone interested in cybersecurity with a comprehensive guide to enhancing SQL server security and utilizing the powerful SQL-MAP tool. Securing SQL servers is a multifaceted undertaking that demands careful consideration of various factors.

The thesis aims to fill existing knowledge gaps and give a deeper understanding of how to identify and prevent SQL injection vulnerabilities. It will introduce readers to the versatility and functionality of the SQL-MAP tool, as it is a key tool in today's cybersecurity environment.

Through practical methods and demonstrations, it aims to provide readers with concrete skills to strengthen SQL server security.

This comprehensive thesis provides a thorough examination of SQL databases, delving into their structure and function. Furthermore, it highlights the consequences of SQL injections and takes a deep dive into the technicalities of TCP/IP, NMAP, and SQL-MAP tools. Along with this technical exploration, the guide advocates for the implementation of more secure and adaptable data management practices in organizations. In the current data-driven landscape, it is crucial to foster a culture that prioritizes safeguarding data, which is essentially the currency of the digital age.

2 SQL Database and SQL Injection

2.1 Structure, Function and History of SQL Databases

Data management, especially when speaking on relational databases, has taken on a widespread form with the development of the Structured Query Language (commonly known as SQL). For understanding the vast infrastructure and nuanced complexity of today's databases, it is important to first go back to their roots.

The origin of SQL: The origin of SQL dates to the early 1970s when IBM researchers, working on System R, one of the first systems to implement Edgar F. Codd's relational database model, developed a new language for data management and retrieval. This language, named SQL, was groundbreaking for its ability to efficiently query data in relational databases. Because of this it quickly became a foundational element in the field of database management, expanding its influence far beyond IBM, eventually being standardized and widely adopted across the industry. (1)

SQL's rise: The adaptability of SQL became clear over the following decades. As digital technology evolved, so did the demands for efficient data management. SQL's capabilities answered to these needs, and it rose to the vanguard of database languages. Today, SQL is the de facto standard for relational database management, making its relevance known in the digital age.

The role of SQL databases in today's organisations: SQL databases today are not just repositories, but dynamic systems fundamentally linked to the structure of organisations. They store, manage, and facilitate access to critical information, acting as both gatekeepers and gateways. Beyond being just storage, these databases play a key role in decision making, analytics and operational efficiency. SQL databases optimise data availability and ensure seamless interaction, giving organisations chance to operate more efficiently.

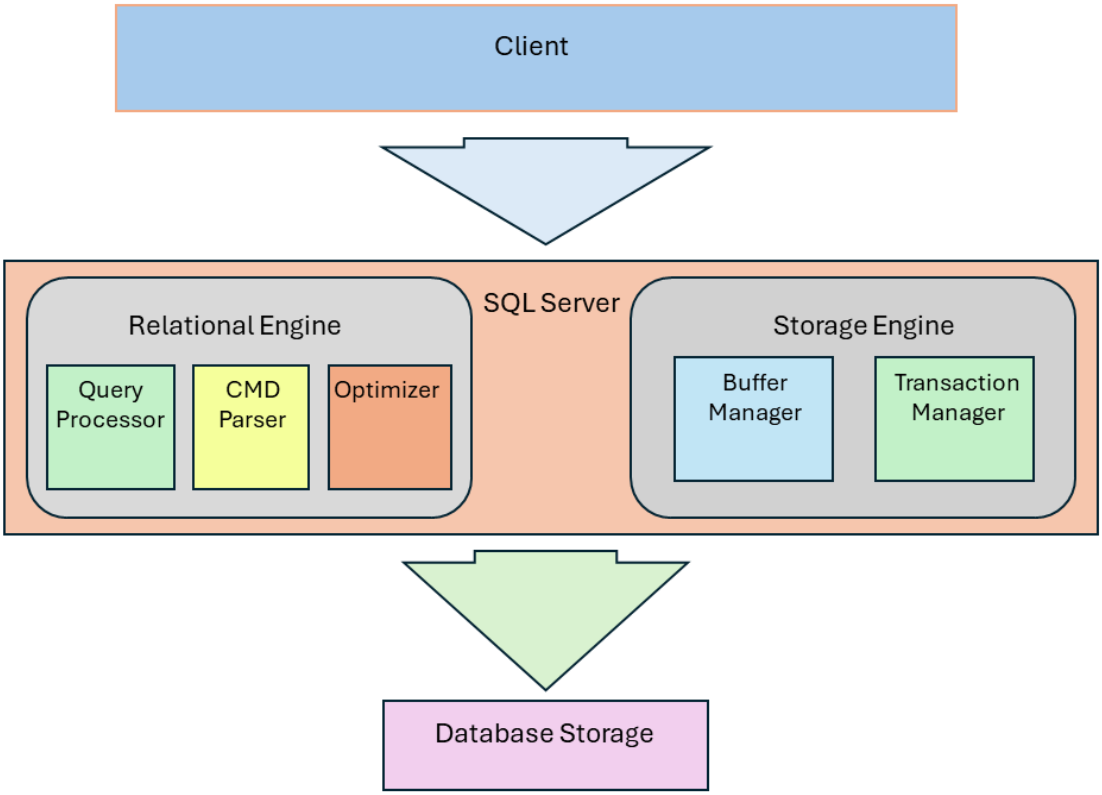

Figure 1: SQL Server Architecture

Figure 1 presents a high-level view of the SQL Server architecture, offering an overview without going into the granular details of each component. Here, we can see the essential layers and components, such as the Client Application, SQL Server, and Database Storage layers. Within the SQL Server layer, we can observe the crucial Relational Engine and Storage Engine, each responsible for critical functions in the database system. While this diagram may simplify certain elements for clarity, it serves as a solid basis for comprehending the fundamental components and interactions within SQL databases. (2)(3)

SQL database architecture: an inherent strength of SQL databases is their relational structure. Tables, or relations, store data in rows and columns, and

each row is identified by unique keys. The relationships between the tables allow the data to be standardized ensuring data integrity and efficiency. In addition, SQL databases use a schema - a predefined plan that determines the structure, types, and relationships between tables. This method of organisation is the key for ensuring data consistency and making complex queries possible. (3)(27)

Functionality: SQL provides a comprehensive set of scripts for various database operations. SQL provides efficient way to work with relational databases, from simple tasks such as retrieving data (SELECT statements) to more complex operations such as joining multiple tables or modifying data (INSERT, UPDATE and DELETE statements). In addition, it supports advanced functions such as transactions and stored procedures, which increases the versatility of databases.

2.2 Evolution of SQL injections and its implications

SQL injections pose a significant and enduring danger to cybersecurity, making it a top priority for protection. Despite the constantly evolving landscape of cyber threats, SQL injections remain a persistent and formidable adversary. This serves as a reminder of the adaptive nature of cyber-attacks and further emphasizes the continual need for fortifying database security. (5)(16)

The Origins of SQL Injection

In the late 1990s and early 2000s, as organizations began to migrate to web-based applications, SQL injection attacks began to become more common. In web-based applications, input data was often not properly validated, allowing malicious SQL commands to be injected and leading to unauthorised access to data or compromised systems. (5)(6)(15)

Evolution of attacks

From early simple attacks, SQL injections have become more complex. Techniques such as Blind SQL Injection, Time-Based Blind SQL Injection and Out-of-Band SQL Injection have increased the diversity of attack methods. While these techniques are based on the basic principle of injecting malicious SQL code, they offer different ways to exploit vulnerabilities and bypass basic defences. (5)(15)(17)

Impacts and Consequences

The consequences of SQL injections are wide ranging. They can lead to unauthorized access to data, exposing everything from user information to company sensitive data. At worst, these attacks can give attackers full control over the targeted systems, leading to data tampering and service disruption. The financial consequences are also significant, including remediation costs, regulatory penalties, and potential litigation. (5)(16)

Mitigation measures

The constant threat of SQL injections has led the cybersecurity community to develop robust defence mechanisms. Key measures include input validation, parameterised queries, and firewalls for web applications. Developer education and awareness are also essential, as many vulnerabilities are caused by poor coding practices. (5)(28)

2.3 Examples of SQL injection attacks

The consequences of SQL injection attacks have been significant and often devastating. Below are examples of real cases that highlight the severity and widespread impact of these vulnerabilities:

Heartland Payment Systems (2008)

In 2008, a devastating cyberattack known as the Heartland Payment Systems breach shook the financial industry. Through a sophisticated SQL injection technique, hackers were able to infiltrate the company's system, compromising a staggering 134 million credit cards. The attack began in 2007, when the perpetrators manipulated the web script code on a login page, granting them access to Heartland's sensitive data. Despite their efforts remaining undetected for months, the consequences were severe. The fraudsters took advantage of the stolen information to create numerous fraudulent credit cards. As a result of this breach, Heartland not only suffered significant financial losses, exceeding $200 million, but also lost the trust of their customers and fell out of compliance with the Payment Card Industry Data Security Standard (PCI DSS). This had a major impact on the company's stock price as well. In 2009, Albert Gonzalez, the mastermind behind the attack, was indicted for his involvement. In the wake of this security.

This incident was a huge lesson for cybersecurity: the importance of transparently disclosing breaches, implementing swift response tactics, ensuring third-party system security, understanding the distinction between compliance and true security, and recognizing the limitations of firewalls in guarding against insider threats. It also serves as a stark reminder of the dangers posed by SQL injection and stresses the critical need for all-encompassing cybersecurity strategies. (4)(7)

Sony Pictures (2011)

In 2011, the world was surprised by a devastating cyberattack on Sony Pictures carried out by the notorious hacker group LulzSec. Their exploitation of a simple SQL injection left over 1 million people's data vulnerable, including personal information such as passwords, email and home addresses, dates of birth, and more. Not only did this breach affect Sony customers, but it also impacted Sony BMG in Belgium and Netherlands, leading to a series of damaging attacks on the company. This event served as a harsh reminder of the dire consequences

of neglecting basic security measures, such as encryption and protection against common cyber threats like SQL injection. It also exposed the glaring security flaws within Sony, resulting in significant financial and reputational losses. (4)(8)(9)

Yahoo! Voices (2012)

In the 2012 Yahoo! Voices hack by D33ds Company, about 450,000 email addresses and passwords were exposed. The hackers used SQL Injection to access Yahoo! Voice servers. One key reason for this was due to the absence of password encryption and HTTPS protocol, which greatly increased the risk of a security breach. This breach affected several domains, such as Gmail, AOL, and Yahoo. To address the issue, Yahoo! acted by fixing the vulnerability and resetting passwords for those affected, while also notifying other potentially impacted companies. This breach was significant because Yahoo! allowed users to register for its Contributor Network using credentials from external sites. It served as a reminder of the importance of implementing strong security measures, including encryption and secure protocols. (4)(10)(11)(12)

TalkTalk (2015)

The 2015 TalkTalk data breach, a major cybersecurity event, involved the British telecom provider TalkTalk. It was executed using an SQL injection, exploiting vulnerabilities from webpages acquired in 2009 from Tiscali's UK operations. Detected on October 21, 2015, due to network slowdowns, TalkTalk swiftly identified and removed the compromised webpages. About 160,000 customers' personal and banking details were illegally accessed, including names, addresses, dates of birth, contact information, and for some, financial details. 15,656 customers had their bank account numbers and sort codes exposed, and partial credit/debit card data of 28,000 customers was also stolen.

The breach cost TalkTalk approximately £77 million. The ICO fined them £400,000 for inadequate security measures, marking it the largest fine by the ICO at that time. Investigations revealed outdated database software and a lack of proactive vulnerability monitoring by TalkTalk. Daniel Kelley, a hacker involved, was sentenced to four years in jail in 2019.

The incident underscores the need for updated security, regular vulnerability monitoring, prompt breach response, and transparent communication, highlighting the significant financial and reputational risks of such breaches. (4)(13)(14)

2.4 Techniques used in SQL Injection

The diversity of SQL injections and their implementation methods are key to security. By understanding these different techniques, more effective defence mechanisms against SQL injection can be developed.

Classical SQL injection

This is one of the most common forms of SQL injection. An attacker directly injects malicious SQL code into the application's input field, causing the database to execute unwanted commands.
This can lead to unauthorized data viewing, manipulation or even execution of system commands. Blind SQL injection unlike classic SQL injection, in Blind SQL injection the attacker does not receive direct feedback on the executed query. Attackers ask the database questions, which can be true or false, and infer the state of the database based on its reactions.

Time-blind SQL injection

In this technique, attackers exploit changes in the application's response time to infer the state of the database. Attackers can intentionally cause delays with SQL commands, allowing them to determine the truth value of their query based on how long it takes to receive a response.

Out-of-band SQL injection
A special case of SQL injection where attackers use a different communication channel to attack and retrieve data. This technique exploits features of databases that allow external connections, allowing data to be transferred over a different channel.

Understanding each technique will help develop targeted defence strategies and improve the overall security of databases. (17)(25)

3 TCP/IP, NMAP, and SQL-MAP

3.1 Overview of TCP/IP and its Role in SQL Server Security

Foundation of TCP/IP
The Transmission Control Protocol/Internet Protocol (TCP/IP) was developed in the 1970s and is the foundation of internet and network connectivity. Its role in the functioning of SQL Server is foundational, making reliable data transfer in varied complex environments possible.
TCP/IP architecture
Consisting of several protocol layers, TCP/IP guides the flow of network traffic and these layers include
The Link Layer manages physical connections and network hardware protocols.
The Internet Protocol (IP) Layer is responsible for addressing packets and routing them across networks.

The Transport Layer (TCP) ensures reliable, error-free transfer of data between applications.
The Application Layer is where network applications use underlying layers for communication. This is the layer we interact with the most.

TCP/IP in SQL Server

The TCP/IP is crucial for ensuring secure data transfers to and from SQL Server. It plays a vital role in managing and safeguarding the server's ports, allowing for the creation of isolated network environments that greatly enhance security.

TCP/IP and Security:

To effectively assess vulnerabilities, a thorough knowledge of TCP/IP is crucial in detecting any unusual network behaviour and potential threats. For SQL Server security, it is highly recommended to secure the commonly used listening port (port 1433) or to consider using a non-standard port, as both measures are significant in maintaining a secure environment.

TLS Integration:

Encrypting data over the network is an important function of SQL Server. In order to achieve this, it utilizes Transport Layer Security (TLS), which exists between the application protocol and TCP/IP layers. TLS employs powerful encryption algorithms, digital certificates, and integrity checks for robust protection. While the inclusion of this feature may add extra steps like network roundtrips and packet encryption/decryption, it is crucial for maintaining secure communication.

Conclusion

A good grasp of TCP/IP and its integration with TLS is important for sturdy SQL Server security. This understanding is important not just in data protection but also in vulnerability assessment, helping in detecting anomalies and potential security breaches.

3.2 Introducing NMAP in the context of SQL Server security

NMAP Overview

NMAP (Network Mapper), is an open-source tool for network discovery and security auditing, was developed by Gordon Lyon (also known as Fyodor Vaskovich) in September 1997. Originally conceived for network exploration and security auditing, NMAP has undergone significant evolution. Today, it has an impressive variety of scanning options (21), sophisticated detection algorithms, and multi-platform support. Within the domain of SQL Server security, the utility of NMAP is especially relevant. The vast amount of features it offers are integral to assessing and pinpointing vulnerabilities SQL Server environments. (18)(19)(20)(22)

NMAP key functions

Host Detection NMAP excels in identifying active devices within a network, a fundamental step in mapping network topology for SQL Server environments.

Port scanning: It provides detailed list of open ports on target hosts. Given that SQL Servers typically listen on specific ports (defaulting to 1433), this feature is indispensable for security assessments.

Version detection: NMAP is adept at determining service and application versions on detected ports, aiding in the identification of potential vulnerabilities in SQL Server instances.

Scripted Interaction: The tool's scripting engine (NSE) allows for the creation and execution of scripts, facilitating a range of network tasks, including advanced security audits suitable to SQL Server.

The role of NMAP in SQL Server security

NMAP stands as a vital tool for professionals safeguarding SQL Server environments. Its applications include:

Security Audit: Using NMAP to scan SQL Server ports and services helps in pinpointing misconfigurations, unpatched services, and vulnerable endpoints.

Penetration testing: It is useful in simulating attacks on SQL Server, identifying potential breaches in security.

Network inventory: NMAP helps in consistently monitoring all devices and services within the SQL Server network environment.

Conclusions

NMAP plays a vital role in establishing a thorough and secure environment, particularly in identifying potential SQL Server vulnerabilities. Although it does not solely focus on SQL injections, its integration with tools like SQL Map provides a powerful strategy to bolster SQL Server security.

3.3 SQL-Map and its application to SQL Server security

Introduction to SQL Map’s functionality

SQL-Map is a tool known for its abilities in detecting and exploiting SQL injection vulnerabilities, making it an indispensable tool in securing SQL Server databases. This chapter goes into the intricacies of SQL-Map's functionality and goes through its application in protecting SQL Server environments.
(23)(24)(26)

Understanding SQL Map functionality

When it comes to identifying and exploiting SQL injection flaws - one of the biggest threats to database security - SQL-Map is the go-to tool. Not only does it automate the process, but it is also highly efficient at testing for various types of vulnerabilities, such as classical, blind, and time-based injections. With its

impressive capabilities, SQL-Map can even uncover complex vulnerabilities in SQL-Server databases.

SQL-Map is the ultimate tool for detecting and exploiting SQL injection vulnerabilities, a major concern for database security. With its comprehensive capabilities, it tackles classical, blind, and time-based injections, making it a powerful asset in uncovering intricate flaws within SQL Server databases. Its specialization in identifying weaknesses unique to SQL Server environments streamlines the testing process and enhances its effectiveness. (23)

Configuring SQL Map for SQL Server

When effectively configured, SQL-Map becomes a powerful tool for pinpointing security vulnerabilities within SQL Server. With the ability to target specific instances, databases, or tables, this tool streamlines the process of conducting focused security assessments. Additionally, its extensive customization options make it adaptable to various SQL Server configurations, making it a valuable asset for optimizing the detection process in different environments.

Analysis of SQL-Map results

Interpreting SQL-Map's findings is key to understanding the vulnerabilities of SQL Server and their potential implications. The tool provides comprehensive analysis that offers insights to be worked on, aiding in fortifying SQL Server defences against SQL injection threats.

Best practices for using SQL-Map

Ethical use of SQL-Map is important. This includes having the necessary permissions and handling any discovered data with care. Regular application of SQL-Map is recommended as a part of an overarching security strategy, ensuring continuing protection against emerging SQL injection vulnerabilities.

Conclusion**s**

When it comes to identifying and addressing SQL injection vulnerabilities, SQL-Map is an essential tool for SQL Server environments. In this section, we delved into the importance of SQL-Map for database security, setting the groundwork for a thorough investigation in the practical demonstration to follow. (26)

4 Practical Part

4.1 Setup and Configuration

Introduction:

For a controlled and secure environment for the practical part, we can utilize a virtualized setup using VMware, hosting Kali Linux. This isolated environment will ensure that our experiments do not harm real-world systems and provides a realistic stage for the demonstration of SQL Map.

Procedure Overview:

1. VMware Workstation Player Setup: First we need to install and setup our virtualization software for Kali Linux.

2. Kali Linux Installation: Kali Linux is a distribution tailored for cybersecurity; It provides us a lot of useful tools for the demonstrations.

3. Post-Installation Configuration: After Kali Linux is setup we can run needed updates, so everything is up to date.

For readers that are interested in the detailed setup process, please refer to Appendix 1.

4.2 Setting Up DVWA on Kali Linux

Introduction:

For the simulation of real-world web application vulnerabilities, especially SQL injection, we can utilize the Damn Vulnerable Web Application (DVWA) that

provides a controlled platform, intentionally designed with security flaws, making it an ideal for our demonstration without risking real-world systems.

For a detailed walkthrough on setting up DVWA on Kali Linux, please refer to Appendix 2.

Procedure Overview:

DVWA Installation: Downloading and configuring DVWA on the Apache server. This involves accessing the web server's root directory, cloning the DVWA repository using Git, and changing the file permissions appropriately.

Server and Database Configuration:
For the next phase we need to ensure that the Apache server and MySQL database are correctly configured. This is important for making sure DVWA will run smoothly, as it relies on these services for its web interface and backend data storage.

MySQL Root Password Configuration:
Next, we walk through securing the MySQL installation by setting or resetting the root password. This step is essential for database security and successful integration with DVWA

DVWA Database Configuration:
This Step involved navigating to DVWA's configuration directory, renaming and editing the configuration file. We need to do this to set the database connection setting correctly to enable DVWA's functionalities.

Security Level Configuration:

Now our DVWA should be up and running, we can now choose a security level for our demonstration we can just choose low.

4.3 Practical Demonstration using SQL-MAP Objective

This section aims to demonstrate the practical application of SQL-MAP for exploiting SQL injection vulnerabilities within a controlled environment. Using the Damn Vulnerable Web Application (DVWA) hosted on a Kali Linux system, we will go through the process of SQL injection attacks.

For a step-by-step tutorial on conducting SQL injection demonstrations using SQL-MAP in the DVWA environment, see Appendix 3.

Methodology

1. Preparation and Setup:

We will now use the DVWA we setup in the last part.

2. Identification of Vulnerable Points:

The SQL injection vulnerability present in DVWA is a standard input field, designed to query user details based on user ID, it will act as the injection point.

First, we do preliminary manual tests to the injection point to validate the vulnerability, setting the stage for a more advanced exploitation using SQL-MAP.

3. Utilization of SQL-MAP:

We will use SQL-MAP to automate the exploitation process. The tool is configured to interface with DVWA, maintaining session consistency via captured PHPSESSID.

A command is executed to probe the database structure of the web application, revealing the extent to which SQL-MAP can extract database information.

Data Extraction and Analysis:

Next the SQL-MAP outputs are analysed, which should provide us insights into the underlying database structure of DVWA.

The tool's ability to enumerate database names, tables, and even specific data entries is highlighted, showing us the severity of SQL injection vulnerabilities.

4. Security Implications:

The demonstration shows the ease with which malicious entities can exploit SQL injection vulnerabilities.

The practical application of SQL-MAP in this controlled setting confirms its effectiveness in identifying and exploiting SQL injection vulnerabilities.

This demonstration provided us valuable insights into the mechanisms of SQL injection attacks, reinforcing the importance of secure coding practices, input validation, and the use of parameterized queries in web application development.

We will continue using SQL-Map in part 4.5 with more advanced exploitation techniques.

4.4 Detecting the Vulnerable Host with NMAP

We will examine the use of NMAP in this section, a powerful tool for network discovery and security auditing, by identifying hosts vulnerable to SQL injection within a network.

Overview of NMAP:

As introduced before NMAP is a versatile and efficient open-source tool for network scanning and security auditing. It provides us tools like host discovery, service, and operating system detection, which are essential in cybersecurity.

NMAP's Application in Vulnerability Detection:

Our demonstration involved deploying NMAP against a controlled environment with a known vulnerable host running DVWA. The aim was to assess the accessibility of the DVWA server and identify potential entry points for SQL injection attacks.

For detailed instructions on preparing and conducting an NMAP scan to identify vulnerable hosts, please consult Appendix 4.

Findings:

The Nmap scan revealed two open ports on the target host (localhost or 127.0.0.1):

Port 80: Running an Apache HTTP server (version 2.4.58), which is hosting the web application (DVWA in this case).
Port 3306: Running a MySQL database (version 5.5.5-10.11.5-MariaDB-3), which is the backend database for the web application.

Implications

Web Server Vulnerabilities:
The Apache server version (2.4.58) should be checked against known vulnerabilities. Older versions might be susceptible to various attacks, including remote code execution or denial of service.
The default Apache page ("It works") indicates that the server might not be fully configured, potentially leaving it vulnerable to misconfiguration exploits.

Database Exposure:

The MySQL version (5.5.5-10.11.5-MariaDB-3) and its configuration details, such as autocommit status, provide insights into potential vulnerabilities.
An open MySQL port (3306) with native password authentication (mysql_native_password) can be a point of attack if not properly secured. This could lead to unauthorized database access, data leakage, or database manipulation.

4.5: Exploiting Vulnerabilities Using SQL-Map

Introduction:

In this part we will build upon what we learned in the part 4.3 basic demonstration by going into more advanced exploitation techniques

Exploitation Techniques:

SQL-Map simplifies the detection and exploitation of SQL injection vulnerabilities by automating the process. Its advanced capabilities allow for complex attacks, such as uncovering key information about the database structure, escalating user privileges, and extracting sensitive data like login credentials and personal information. By inputting specific commands, we are able to retrieve a plethora of details, including database names, table names, and column names. This tool is essential for comprehensive database security.

Security Implications:

The ease with which SQL-Map can extract sensitive information from a vulnerable system is almost disturbing. This serves as a crucial reminder of the danger posed by potential data breaches, putting confidential information at risk and compromising user privacy. It is imperative for web applications to have rigorous security measures in place, which includes frequent vulnerability scans and prompt remediation of any issues.

Conclusion:

The results of this exploitation using SQL-Map demonstrate the tool's effectiveness in uncovering significant security vulnerabilities. This further highlights the pressing need for organizations to prioritize strong web application security measures, such as secure coding practices, frequent security audits, and ongoing monitoring for emerging vulnerabilities.

For an in-depth guide to exploiting SQL injection vulnerabilities using SQL-Map, including advanced techniques, refer to Appendix 5.

4.6 Protective Measures Against SQL Injection Attacks

Understanding SQL Injection:

SQL injection continues to pose a significant threat to the security of databases, especially in SQL Server environments. Its impact can range from unauthorized access to data, to full-scale system takeover. However, the recognition and usage of tools like SQL Map not only helps detect vulnerabilities, but also emphasizes the importance of implementing secure coding practices.

Defensive Strategies in SQL Server:

In order to effectively defend against SQL injection, it is crucial to implement best practices in SQL Server management. This includes utilizing parameterized queries and prepared statements, which are crucial in preventing

the execution of malicious code. Regular vulnerability scanning, such as with SQL Map, is also essential in identifying and resolving any potential weaknesses."

Security Audits and Continuous Monitoring:

To preserve the integrity of SQL Server databases, it is imperative to conduct ongoing security audits and maintain vigilant monitoring. While conducting regular assessments, such as leveraging SQL Map for vulnerability scanning, is a critical step in detecting vulnerabilities, it should not be the only approach. A robust security plan should encompass a range of tools and practices to ensure comprehensive protection. (15)(16)

For detailed strategies and procedures on implementing protective measures against SQL injection attacks, see Appendix 6.

4.7 Summary of Practical Demonstration

Overview of Demonstrations:

The practical demonstrations conducted in Section 4 aimed to expose the vulnerabilities inherent in SQL Server and demonstrate the efficacy of SQL-Map in identifying and exploiting these vulnerabilities. This process was crucial for understanding the real-world implications of SQL injection attacks and the necessity for robust defense mechanisms.

Key Findings from SQL-Map Demonstrations:

Vulnerability Identification: SQL-Map proved to be a powerful tool in uncovering SQL injection vulnerabilities. Its ability to automate the detection process highlighted the importance of employing such tools in regular security audits.

Data Extraction Capabilities: The tool's efficiency in extracting sensitive information from the database was alarming. It underscored the potential risks associated with unsecured SQL databases and the ease with which attackers could access confidential data.

Exploitation of Vulnerabilities: The practical use of SQL-Map

demonstrated how quickly and effortlessly SQL injection vulnerabilities could be exploited, leading to unauthorized data access and potentially severe security breaches.

Protective Measures Against SQL Injection:

The demonstrations emphasized the critical role of input validation and parameterized queries in safeguarding against SQL injection attacks. Regular vulnerability scans, specifically utilizing tools such as SQL-Map, were deemed crucial in promptly detecting and addressing potential threats. Furthermore, the importance of raising awareness and providing training on secure coding practices and database management was emphasized. This includes educating developers and database administrators on common vulnerabilities and effective prevention strategies.

Conclusion:

The practical demonstrations using SQL-Map served as a valuable learning experience, portraying the risks that SQL injection attacks present and SQL-Map tools effectiveness for identifying the vulnerabilities. As SQL Server continues to evolve, it is imperative to stay vigilant and adopt a proactive approach to security. This involves regularly updating systems, implementing best security practices, and leveraging advanced tools such as SQL-Map for ongoing surveillance. These interactive demonstrations should serve as a

wake-up call for organizations to strengthen their database security and protect their critical data from potential cyberattacks.

5 Summary

Overview

My thesis, "Enhancing SQL Server Security with the Utilization of SQL-Map," offers a thorough investigation into the intricate elements of SQL Server security. It stresses the vital importance of utilizing tools such as SQL-Map in safeguarding databases. By delving into both theoretical concepts and practical implementations, this research illuminates the underlying vulnerabilities of SQL databases and highlights the effectiveness of SQL-Map in detecting and minimizing these potential threats.

Key Takeaways

The vulnerability of SQL Servers leaves them exposed to a host of security threats. Particularly worrisome is the risk of SQL injection, which can lead to unauthorized access, data theft, and even a complete system takeover. However, amidst these concerns, SQL-Map stands out as a valuable resource for both penetration testers and database administrators. With its advanced ability to recognize and exploit SQL injection vulnerabilities, it is an indispensable asset for any robust security strategy.

Practical Demonstrations: The usage of SQL-Map in hands-on demonstrations gave us a better understanding of the mechanics behind SQL injection attacks, as well as effective prevention methods. Through these activities, we were able to witness the tool's effectiveness in detecting vulnerabilities, retrieving information, and taking advantage of weaknesses. Furthermore, my thesis has emphasized the crucial role of stringent security measures in keeping SQL

Server databases safe from potential attacks. This includes implementing input validation, utilizing parameterized queries, regularly conducting vulnerability scans, and strictly following industry best practices. By implementing these measures, we can effectively safeguard our databases from any potential threats.

Future outlook: In today's rapidly-changing cybersecurity landscape, the significance of tools such as SQL-Map for safeguarding SQL Server cannot be overstated. It is essential to keep up with frequent updates, conduct routine security evaluations, and stay informed about emerging threats and advancements in order to maintain strong database security.

Concluding Thoughts

This thesis highlights the importance of taking a proactive and all-encompassing approach to SQL Server security. Utilizing tools such as SQL-Map, in conjunction with implementing strong security measures, is imperative in keeping sensitive data safe and keeping the integrity of SQL Server databases. The findings of this research provide valuable guidance to both organizations and individuals in bolstering their database defenses against the constantly evolving realm of cyber threats. It is important to note that ensuring the security of SQL Server is an ongoing task that demands diligence, awareness, and the proper tools for effectively managing and mitigating risks.

References

1. Early History of SQL. 2012. https://ieeexplore.ieee.org/document/6359709 Accessed 25.9.2024
2. Relational Engine. 2014. https://www.sqlservergeeks.com/sql-server-architecture-part-2-the-relational-engine/ Accessed 25.9.2024
3. SQL Server Architecture. 2023. https://www.guru99.com/sql-server-architecture.html. Accessed 5.12.2023
4. SQL Injection Examples. 2024. https://softwarelab.org/blog/sql-injection-examples/ Accessed 19.9.2023
5. SQL Injection Vulnerability History. 2013. https://www.invicti.com/blog/web-security/sql-injection-vulnerability-history/ Accessed 19.9.2023
6. How was SQL Injection discovered. 2013. https://www.esecurityplanet.com/networks/how-was-sql-injection-discovered/ Accessed 19.9.2023
7. Hearthland Data Breach. 2015. https://www.proofpoint.com/us/blog/insider-threat-management/throwback-thursday-lessons-learned-2008-heartland-breach Accessed 20.9.2023
8. Sony Pictures Hack. 2014. https://www.trendmicro.com/vinfo/us/security/news/cyber-attacks/the-hack-of-sony-pictures-what-you-need-to-know Accessed 20.9.2023
9. Sony Pictures Hack. 2015. https://www.forbes.com/sites/natalierobehmed/2015/04/16/the-entire-sony-hack-is-now-available-on-wikileaks/ Accessed 20.9.2023
10. Yahoo Voices Data Breach. 2012. https://www.pcmag.com/archive/yahoo-voices-breach-exposes-453000-passwords-300180 Accessed 15.10.2023
11. Yahoo Voices Data Breach. 2012. https://www.darknet.org.uk/2012/07/yahoo-voices-hacked-with-sql-injection-passwords-in-plaintext/ Accessed 15.10.2023
12. Yahoo Voices Data Breach. 2012. https://money.cnn.com/2012/07/12/technology/yahoo-hack/ Accessed 16.10.2023
13. TalkTalk Cyberattack. 2015. https://ico.org.uk/about-the-ico/media-centre/talktalk-cyber-attack-how-the-ico-investigation-unfolded/ Accessed 16.10.2023
14. TalkTalk CyberAttack. 2015. https://tripwire.com/state-of-security/the-talktalk-breach-timeline-of-a-hack Accessed 16.10.2023
15. SQL Injection. 2023. https://owasp.org/www-community/attacks/SQL_Injection Accessed 8.11.2023
16. What is an SQL Injection. No Date. https://portswigger.net/web-security/sql-injection Accessed 7.11.2023
17. SQL Injection CheatSheet. No Date. https://www.invicti.com/blog/web-security/sql-injection-cheat-sheet/ Accessed 19.9.2023
18. Nmap. https://nmap.org/ Accessed 10.12.2023
19. Nmap Reference Guide. https://nmap.org/book/man.html Accessed 10.12.2023
20. Nmap Port Scanning Tutorial https://nmap.org/book/port-scanning-tutorial.html Accessed 10.12.2023

21. Nmap Options Summary. https://nmap.org/book/man-briefoptions.html Accessed 10.12.2023
22. About Nmap Creator. https://insecure.org/fyodor/ Accessed 10.12.2023
23. SqlMap https://github.com/sqlmapproject/sqlmap/wiki/Introduction Accessed 14.9.2023
24. SqlMap https://sqlmap.org/ Accessed 14.9.2023
25. SqlMap Techniques https://github.com/sqlmapproject/sqlmap/wiki/Techniques Accessed 15.9.2023
26. SqlMap History https://github.com/sqlmapproject/sqlmap/wiki/History Accessed 15.9.2023
27. Sql Server Best Practices. 2023. https://learn.microsoft.com/en-us/sql/relational-databases/security/sql-server-security-best-practices?view=sql-server-ver16 Accessed 5.1.2024
28. DVWA sql injection. 2023 https://medium.com/@hashsleuth.info/how-to-exploit-dvwa-blind-sql-injection-sqli-with-sqlmap-and-burp-suite-e4b3f08a0dfc Accessed 15.12.2023
29. Damn Vulnerable Web Application https://github.com/digininja/DVWA Accessed 15.12.2023

Appendices

Appendix 1: Setting Up the Virtual Environment

Installing VMware Workstation Player:

- Download and Install VMware:
  - Navigate to the VMware download page: VMware Workstation Player
  - Download the version for your OS (Windows/Linux).
  - Run the installer and follow the prompts.
  - Launch VMware Workstation Player after installation.

Downloading Kali Linux ISO:

- Obtain Kali Linux Image:
  - Visit the Kali Linux downloads page: Kali Downloads
  - Download the Kali Linux 64-bit or 32-bit ISO.

Creating a New Virtual Machine:

- VM Setup in VMware:
  - Open VMware and choose "Create a New Virtual Machine."
  - Select "Installer disc image file (iso)" and locate the Kali ISO.
  - Set the guest OS as "Linux" and version as "Debian 10.x 64-bit" or "32-bit."
  - Name the VM (e.g., "Kali Linux VM").
  - Allocate disk space (20 GB recommended) and store as a single file.
  - Customize hardware: Allocate at least 2 GB RAM and 2 CPU cores.
  - Complete the setup.

Installing Kali Linux on the VM

Kali Installation Process:

Start the VM and select "Graphical Install."

Choose language, location, and keyboard layout.

Set a hostname (e.g., "kali") and root password.

Select time zone, partitioning method (guided), and write changes.

Reboot after installation and eject installation media if prompted.

Boot into Kali Linux desktop.

Post-Installation Configuration

Kali Linux Initial Setup:

Open terminal in Kali Linux.

Update and upgrade packages: sudo apt update && sudo apt upgrade -y

Reboot system to apply updates.

Appendix 2: Detailed Procedure for Setting Up DVWA on Kali Linux

Server and Database Configuration:

- Ensure the Apache and MySQL Services are Running:
  - Start Apache: sudo service apache2 start
  - Start MySQL: sudo service mysql start

DVWA Installation:

- Navigate to the Web Server's Root Directory:
  - cd /var/www/html/
- Download DVWA:
  - sudo git clone https://github.com/digininja/DVWA.git
- Change Permissions for DVWA Directory:
  - sudo chown -R www-data:www-data DVWA/

MySQL Root Password Configuration:

- Access MySQL as Root:
  - sudo mysql
- Set or Reset the MySQL Root Password:
  - For newer versions: ALTER USER 'root'@'localhost' IDENTIFIED BY 'your_new_password';
  - For older versions: SET PASSWORD FOR 'root'@'localhost' = PASSWORD('your_new_password');
- Flush Privileges and Exit MySQL:
  - FLUSH PRIVILEGES;
  - EXIT;

DVWA Database Configuration:

- Navigate to DVWA's Config Directory:
  - cd /var/www/html/DVWA/config/
- Rename the Configuration File:

sudo mv config.inc.php.dist config.inc.php

Edit the Configuration File:

Use a text editor like nano: sudo nano config.inc.php

Update the *DB_PASSWORD* line with your MySQL root password.

Save and Exit the Editor.

Accessing DVWA:

Open a Web Browser and Navigate to DVWA:
"http://localhost/DVWA/"

Initialize the Database:
Click on "Create / Reset Database".

Log in with Default Credentials:
Username: admin
Password: password

Security Level Configuration:

Adjust DVWA Security Level:
Click on "DVWA Security" and set the level to "Low" for our demonstrations.

Appendix 3: SQL-MAP Practical Demonstration Tutorial

In this appendix, you will find a comprehensive walkthrough for conducting an initial test with SQL-MAP in the Damn Vulnerable Web Application (DVWA) environment. It is assumed that DVWA is up and running on a Kali Linux system.

Initial SQL-MAP Test

Launch SQL-MAP:

Open a terminal in Kali Linux and prepare to launch SQL-MAP.

Configure SQL-MAP for DVWA:

Ensure that you have the PHPSESSID from your DVWA session.

Execute Initial Test Command:

Run the following SQL-MAP command:

```
sqlmap –u "http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="security=low; PHPSESSID=your_session_id"
```

Objective: This command tests the SQL injection vulnerability in DVWA and confirms that SQL-MAP is properly interacting with the web application.

Note: Replace "your_session_id" with the actual PHPSESSID value from your DVWA session.

Review Output:

Examine the results generated by SQL-MAP to verify the existence of the SQL injection vulnerability. This crucial assessment lays the groundwork for more sophisticated methods of exploiting SQL injection, which will be explored in greater depth later on.

Appendix 4: Procedure for Detecting Vulnerable Host with NMAP

Preparing the Environment:

Ensure DVWA is installed and running on the target host.

Confirm NMAP installation on your system or install it via sudo apt-get install nmap.

Determine your network range.

Checking Your Network Range:

Open a terminal.

Type ip a or ifconfig to display network interfaces and IP addresses.

Identify your active network interface (e.g., eth0, wlan0) and note the IP address.

Determine the network range based on your IP address and subnet mask. For most home networks, the range is typically 192.168.x.x/24 or 10.0.x.x/24.

Conducting the NMAP Scan:

Run a basic NMAP scan to find active hosts: nmap -sn your-network-range (replace your-network-range with your actual network range).

Execute a detailed scan on the DVWA host: nmap -sV -A DVWA-host-IP (replace DVWA-host-IP with the actual IP of your DVWA host).Scan Results:

Host: localhost (127.0.0.1)

Open Ports:

Port 80 (HTTP):

Service: Apache httpd 2.4.58 (Debian)

Notes: Default Apache page served.

Port 3306 (MySQL):

Service: MySQL 5.5.5-10.11.5-MariaDB-3

Features: Autocommit enabled, various capabilities flags indicating transaction support, compression, multiple statement, and result support, among others.

Authentication Plugin: mysql_native_password

Interpretation:

The open HTTP port suggests that the web application (DVWA) is hosted on this server. The server's configuration and version should be examined for potential vulnerabilities.

The MySQL service on port 3306 highlights a database that is integral to the web application. The version and configuration settings indicate areas that need to be secured to prevent SQL injection attacks or unauthorized access.

Security Recommendations:

Regularly update the Apache server and MySQL database to the latest versions to mitigate known vulnerabilities.

Configure the web server properly to avoid default settings that could be exploited.

Secure the MySQL database by implementing strong password policies and considering advanced authentication mechanisms.

Regularly conduct security assessments and penetration testing to identify and mitigate potential vulnerabilities.

Appendix 5: Exploiting SQL Injection Vulnerabilities Using SQL-Map

Step-by-Step Guide for Advanced Exploitation with SQL-Map

Starting SQL-Map:

Open the terminal and ensure SQL-Map is ready for use.

Enumerating Databases:

Command: sqlmap -u "http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="PHPSESSID=your_session_id; security=low" --dbs

Purpose: Lists all databases available in the web application. You should see dvwa among others.

Listing Tables in the DVWA Database:

Command: sqlmap -u "http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="PHPSESSID=your_session_id; security=low" -D dvwa --tables

Purpose: Lists all tables in the dvwa database.

Extracting Data from a Specific Table:

After identifying the tables in the dvwa database, choose a table to target (e.g., users).

Command: sqlmap -u "http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="PHPSESSID=your_session_id; security=low" -D dvwa -T users --columns

This command lists all columns in the users table.

Dumping Data from Specific Columns:

Choose columns to extract data from (e.g., user, password).

Command: sqlmap -u "http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="PHPSESSID=your_session_id; security=low" -D dvwa -T users -C user,password --dump

This command extracts data from the specified columns in the users table.

Exploring Advanced SQL-Map Options:

Utilize SQL-Map's advanced features like --risk, --level, and specific payloads for different types of SQL injections (e.g., --technique=B for Boolean-based blind SQL injection).
Command Example: sqlmap -u "http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="PHPSESSID=your_session_id; security=low" -D dvwa --risk=3 --level=5 --technique=B
This command configures SQL-Map to use a higher risk and level setting with a specific technique, providing a more aggressive and thorough examination.

Interpreting the Output:

Review the data retrieved by SQL-Map, focusing on the sensitivity and type of data exposed.
Analyse the output for potential vulnerabilities, such as weak password hashes or the exposure of personal data.

Securing the Application

Immediate Actions:

Implement parameterized queries and prepared statements to mitigate SQL injection risks.
Regularly update and patch the database management system and web application frameworks.

Long-Term Strategies:

Conduct regular security audits and penetration tests to identify and fix vulnerabilities.

Emphasizing the importance of secure coding practices. So, the future projects will be safer than ever.

Reflecting on Findings:

Discuss how the demonstration with SQL-Map highlights the ease with which a poorly secured database can be compromised.

Emphasize the importance of robust database security measures and the ongoing need for vigilance against evolving threats.

Note: It is crucial to replace "your_session_id" with the actual PHPSESSID value from your DVWA session. Also, adapt the commands based on the specific requirements and configurations of your environment.

Appendix 6: Implementing Protective Measures

Using SQL Map for Regular Vulnerability Scanning:

Schedule Regular Scans: Set up a routine (e.g., weekly, or monthly) to run SQL Map scans against your SQL Server databases. This ensures ongoing monitoring of potential vulnerabilities.

Command: sqlmap -u "target-URL" --batch --dbms="mssql"

Replace "target-URL" with the specific URL of the web application connected to your SQL Server.

Scan for Specific Vulnerabilities: Focus SQL Map scans on types of SQL injection vulnerabilities relevant to SQL Server.

Example: sqlmap -u "target-URL" --risk=3 --level=5 --dbms="mssql"

The --risk and --level flags increase the depth and breadth of the scan.

Identify Database Structure: Use SQL Map to understand the structure of the database, which is crucial for comprehending potential attack vectors.

Command: sqlmap -u "target-URL" --dbms="mssql" --dbs

This command lists all databases in the SQL Server.

Test Specific Database Parameters: Conduct targeted tests on parameters that are more likely to be vulnerable.

Command: sqlmap -u "target-URL" -p "parameter" --dbms="mssql"

Replace "parameter" with the specific query parameter you want to test.

Automate and Integrate Scanning: Automate SQL Map scans using cron jobs or integrate them into your CI/CD pipeline for continuous security assessment.

Post-Scan Actions:

Once the SQL Map scan is completed, it is important to thoroughly analyse the results for any signs of vulnerability or potential weaknesses within your SQL Server. This crucial step enables the identification of critical areas that require immediate attention to strengthen the overall security of your system. Based on these findings, swift action must be taken to fortify the security of your system, such as applying necessary patches, updating software, and revising database access controls to

safeguard sensitive data. Additionally, keeping detailed records of the scan results and remediation steps is vital for monitoring security progress and planning future protective measures. In essence, documenting and reporting these findings promotes a proactive and secure approach towards protecting your system.