UNIVERSITÀ DEGLI STUDI DI MILANO
FACOLTÀ DI SCIENZE E TECNOLOGIE

DIPARTIMENTO DI INFORMATICA
GIOVANNI DEGLI ANTONI



Corso di Laurea triennale in
Sicurezza dei Sistemi e delle Reti Informatiche

Microsoft Azure Database Protection against Cyber Attacks

Relatore: **Prof.ssa Pierangela SAMARATI**
Correlatore: **Ing. Viktor TSYKUNOV**

Tesi di Laurea di:
Sergio MELONI
Matr. 773779

ANNO ACCADEMICO 2022/2023

Ringraziamenti / Acknowledgements

Questa tesi é dedicata soprattutto alla mia famiglia. É il frutto non solo del mio impegno ma del sostegno incessante di chi mi vuole bene. É dedicata alla mia mamma, che é sempre nei nostri cuori. É dedicata al mio papá che mi é sempre stato vicino, incoraggiandomi e guidandomi per tutto il percorso, e non facendomi mai pesare i momenti di sconforto. Grazie per non avermi mai fatto mancare niente e avermi insegnato a rischiare, volere sempre di piú e non arrendermi. É dedicata a mia zia Daniela e mio zio Virgilio. Lo sapete, siete i miei secondi genitori e non vi ringrazieró mai abbastanza per tutto. E infine, un ringraziamento a me stesso, per aver sempre lottato con determinazione verso il raggiungimento di questo traguardo. Questa tesi non segna solo la fine di un capitolo, ma l'inizio di un nuovo viaggio.

This thesis is dedicated above all to my family. It is the fruit not only of my own effort but of the incessant support of those who care about me. It is dedicated to my mom, who is always in our hearts. It is dedicated to my dad, who has always been by my side, encouraging me and guiding me throughout this journey, and never letting the moments of despair weigh me down. Thank you for never letting me lack anything and teaching me to take risks, to always want more, and to never give up. It is dedicated to my aunt Daniela and my uncle Virgilio. You know, you are my second parents, and I will never be able to thank you enough for everything. And finally, a thank you to myself, for always fighting with determination towards achieving this goal. This thesis marks not just the end of a chapter, but the beginning of a new journey.

Sergio

Contents

Ringraziamenti / Acknowledgements **i**

Index **ii**

1 Cloud Computing Foundations and Azure Insights **3**
1.1 Cloud Computing and Players . . . 3
1.2 Microsoft Azure . . . 5
1.3 Compliance and Regulatory Standards . . . 6
1.4 Data Sovereignty and Localization . . . 7
1.4.1 Microsoft Cloud for Sovereignty . . . 8
1.5 Azure Governance and Cost Management . . . 9

2 Evolution and Diversity of Database Systems **11**
2.1 Historical Introduction to Databases . . . 11
2.2 Diversity of Database Models . . . 12
2.2.1 Relational Database Systems . . . 12
2.2.2 NoSQL Databases . . . 14
2.3 Present Time . . . 15
2.4 Going ahead: NewSQL . . . 16
2.4.1 NewSQL Databases . . . 17

3 Microsoft Azure for Database Management **18**
3.1 Relational Databases . . . 18
3.1.1 Azure SQL . . . 18
3.1.2 Azure Database for MySQL, PostgreSQL, and MariaDB . . . 20
3.2 Non-Relational Databases . . . 21
3.2.1 Cosmos DB . . . 21
3.2.2 Azure Cache for Redis . . . 22
3.2.3 Azure Synapse Analytics . . . 23
3.2.4 Azure Table Storage . . . 24

4 Data Security and Privacy Principles **26**
4.1 Foundational Principles of Data Security . . . 26
4.1.1 Confidentiality . . . 26
4.1.2 Integrity . . . 27
4.1.3 Availability . . . 27
4.2 Privacy Principles and Compliance . . . 27
4.2.1 Data Minimization . . . 28
4.2.2 Purpose Limitation . . . 28
4.2.3 Consent . . . 28

4.2.4 Compliance Frameworks 28
4.3 Data Security Challenges in Databases 29
4.3.1 Vulnerability Management 29
4.3.2 Data Encryption 30
4.3.3 Access Control 30
4.3.4 Data Masking and Tokenization 30
4.4 Implementing Privacy by Design in Azure Deployments 31
4.4.1 Privacy Impact Assessments (PIAs) 31
4.4.2 Data Classification 31
4.4.3 Least Privilege Access 31
4.5 Best Practices for Secure Database Deployment on Azure 32
4.5.1 Security Baseline Configuration 32
4.5.2 Regular Auditing and Monitoring 32
4.5.3 Data Recovery and Backup 33

5 Database Threats **34**
5.1 SQL Injection 34
5.1.1 Consequences of a SQL Injection Attack 35
5.1.2 Types of Attack 35
5.1.3 Typical attack 36
5.1.4 Mitigating SQL Injection Vulnerabilities 37
5.2 Denial of Service (DoS) Attacks 37
5.2.1 Types of DoS Attacks 37
5.2.2 Impact of DoS Attacks 38
5.2.3 Mitigation Strategies 39
5.3 Insider Threats 39
5.3.1 Malware and Ransomware 40
5.3.2 Defense Mechanisms 40
5.4 Data Leakage and Unauthorized Access 41
5.4.1 Exposure through APIs 41
5.4.2 Poor Authentication Mechanisms 41
5.4.3 Lack of Data Encryption 41
5.5 Configuration and Operational Flaws 42
5.5.1 Default Settings 42
5.5.2 Inadequate Patch Management 42
5.5.3 Improper Access Controls 43

6 Microsoft Azure's Defensive Strategies for Database Security **44**
6.1 Azure Firewall 44
6.1.1 Implementing Azure Firewall for Database Protection 45
6.2 Identity Management: Fortifying Access Control 45
6.2.1 Multi-Factor Authentication (MFA) 46
6.2.2 Conditional Access Policies 46
6.2.3 Role-Based Access Control (RBAC) 47
6.3 Intrusion Detection and Prevention Systems (IDPS) 48
6.3.1 Key Components of Azure's IDPS 49
6.3.2 Implementing Azure IDPS for Database Security 49
6.3.3 Machine Learning in Azure Threat Detection 50

7 Protection in practice: the development of a secure .NET Library 52
7.1 The project: DotNetArmor . . . 52
7.2 Library Specifics . . . 53
7.3 Secret Management . . . 54
7.4 Secure Password Management . . . 56
7.5 Hashing Manager . . . 58
7.6 Symmetric Cryptography . . . 59
7.7 Asymmetric Cryptgraphy . . . 60
7.8 Input Sanitization . . . 61
7.9 Secret Sharing Management . . . 62
7.10 AntiDOS and Rate Limiting API . . . 63

8 Machine Learning in Threat Detection: A concrete approach 65
8.1 Machine Learning tools for enhancing Security . . . 65
8.2 Azure Machine Learning . . . 66
8.3 Azure AutoML . . . 66
8.3.1 Cleaning and Preprocessing steps . . . 67
8.3.2 Model Training . . . 67
8.3.3 ML Algorithms . . . 68
8.3.4 Voting Ensemble . . . 69
8.3.5 Model Registration . . . 70
8.3.6 Endpoint Deployment . . . 70
8.4 Improving DB Threat Detection in DotNetArmor . . . 71
8.4.1 Challenge: Overusage of the REST ML API . . . 71
8.4.2 Challenge: Overhead from Repeated Queries . . . 71
8.4.3 Challenge: Model Accuracy and False Positives/Negatives . . . 72
8.4.4 Challenge: Keeping Up with Evolving Threats . . . 73
8.5 Testing the endpoint . . . 73
8.5.1 Testing Methodology . . . 73
8.6 Future Improvements . . . 74
8.6.1 Autolearning and Continuous Improvement . . . 74
8.6.2 Enhancing Model Retraining . . . 74

A Source Code 76
A.1 Service Registration . . . 76
A.2 Services . . . 77
A.2.1 AzureKeyVaultService . . . 77
A.2.2 SQLInjectionDetectionService . . . 78
A.2.3 DetectSqlThreatMiddleware . . . 79
A.2.4 PasswordService and *Have I Been Pwned?* Service . . . 80
A.2.5 SqlService . . . 83
A.2.6 Input validation example: Password Strength Attribute . . . 84

Abstract

In the current digital landscape, an increasing number of companies are turning to cloud solutions to manage their data and, more importantly, their customers' data. Data security has thus become of extreme importance, as data now serves as the key to unlocking market opportunities and offering more sophisticated services. Concurrently, we are witnessing the rapid growth of cloud providers, who are continually expanding their range of services. In this vast sea of information, small and medium-sized enterprises (SMEs) that are navigating cloud providers for the first time may find it challenging to discern which services to prioritize.

This thesis aims to serve as an introductory guide for businesses venturing into the realm of cloud data protection, with a particular focus on Microsoft Azure. Through theoretical exploration, it will introduce the various database services available on the platform, highlighting their differences, significance, and the advantages and disadvantages of each product. Additionally, it will outline the obligations companies must adhere to regarding the protection of personal data, as well as the services Azure offers to ensure compliance. The thesis will also discuss the main threats to databases and how attackers can exploit data for malicious purposes, alongside Azure techniques and services designed for protection.

Furthermore, the theoretical framework is bolstered by a practical project involving the creation of a library for web projects that guards against the number one database attack: SQL Injection. This is achieved through the use of techniques provided by Microsoft frameworks and machine learning. This represents an additional tool for companies newly engaging with Azure cloud services for data and database threat protection, which they may initially find costly and prohibitive.

Thesis Organization

- Acknowledgments
- Contents
- Abstract
- **Chapter 1**: Cloud Computing Foundations and Azure Insights - This chapter provides an overview of cloud computing, its benefits and challenges, followed by a detailed examination of Azure and its prominence in the cloud market.
- **Chapter 2**: Evolution and Diversity of Database Systems - Discusses the historical development of database management systems, the different types (relational, NoSQL, etc.), and an overview of the most prevalent databases in use today.
- **Chapter 3**: Microsoft Azure for Database Management - Describes the range of database services offered by Azure, including Azure SQL Database and Cosmos DB, and their unique features.
- **Chapter 4**: Data Security and Privacy Principles - Explores the fundamental concepts of data security, such as encryption and backups, and discusses how these principles are applied in cloud and Azure environments.
- **Chapter 5**: Database Threats - Details common threats to databases, such as SQL injection and cross-site scripting, and discusses the evolving nature of these threats.
- **Chapter 6**: Microsoft Azure's Defensive Strategies for Database Security - Delves into Azure's specific security solutions for protecting databases, including firewalls, intrusion detection and prevention systems, and identity management.
- **Chapter 7**: Protection in practice: the development of a secure .NET Library - Describes the creation of a .NET library - DotNetArmor - specifically designed for developers and SMEs exploring the .NET world for the first time and in need of a fast, secure onboarding.
- **Chapter 8**: Machine Learning in Threat Detection - The chapter demonstrates how to create a REST endpoint for SQL Injection detection using Azure Machine Learning, an API that will be added to the DotNetArmor library to enhance security and provide a tool to counteract the number one attack on databases.
- References and Bibliography
- Appendix: Source Code

Chapter 1

Cloud Computing Foundations and Azure Insights

1.1 Cloud Computing and Players

Cloud computing has revolutionized the way businesses and individuals store, process, and access data, delivering unparalleled scalability, flexibility, and cost-efficiency. Born out of the need to overcome traditional IT infrastructure limitations, cloud computing has rapidly evolved over the years, transforming the technological landscape and enabling organizations to leverage computing resources on-demand [52]. This comprehensive introduction explores the origins of cloud computing, its historical development, the current state of major cloud providers, and future trends, while delving into the advantages and disadvantages of this paradigm-shifting technology.

The concept of cloud computing emerged as a solution to address the limitations and challenges posed by traditional IT infrastructure. In the past, organizations had to invest heavily in on-premises hardware and software, resulting in high capital expenditures and maintenance costs. Furthermore, scaling resources to meet fluctuating demands often proved to be a time-consuming and expensive process. Cloud computing was conceived as a way to overcome these limitations by providing a scalable, on-demand model for accessing computing resources. The term "cloud computing" itself gained prominence in the early 2000s [65], although the roots of the concept can be traced back further. Early approaches to cloud computing can be seen in the emergence of grid computing and utility computing in the 1990s and early 2000s. Grid computing focused on harnessing distributed computing resources to solve complex problems, while utility computing aimed to provide computing resources on a pay-as-you-go basis, similar to traditional utility services.

The evolution of cloud computing witnessed the development of key technologies and concepts that paved the way for its widespread adoption. Virtualization technology played a crucial role by enabling the abstraction of physical hardware into virtual machines (VMs) [13], allowing for efficient resource utilization and flexibility. This breakthrough technology formed the foundation for the scalability and resource allocation capabilities of cloud computing.

In the early stages, cloud computing primarily targeted infrastructure-level services, known as Infrastructure as a Service (IaaS), which provided virtualized computing resources, storage, and networking capabilities to users. This was followed by the introduction of Platform as a Service (PaaS), which offered a higher level of

abstraction, allowing developers to focus on application development rather than infrastructure management. Finally, Software as a Service (SaaS) emerged, providing ready-to-use software applications accessible via the internet. In the past, organizations faced significant challenges with on-premises infrastructure, such as limited scalability, high maintenance costs, and the need for manual software updates and security patches. Cloud computing offered a paradigm shift by providing organizations with the ability to scale resources up or down based on demand, reducing capital expenses, and leveraging automated updates and security enhancements provided by cloud service providers. [12]

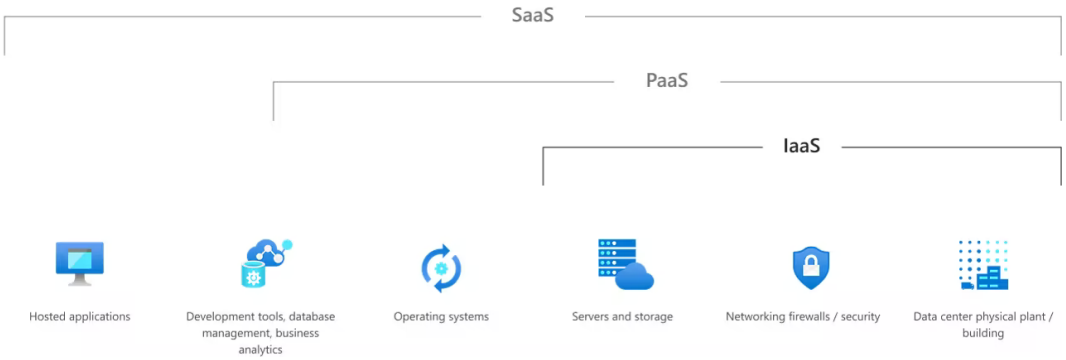


Figure 1: Cloud Deployments
Source: `https://azure.microsoft.com/`

Today, the cloud computing market is dominated by three major providers: according to *STATISTA.com* [58] and *Synergy Research Group*, the top three of Cloud Providers is made by **Amazon Web Services** (AWS), **Microsoft Azure**, and **Google Cloud Platform** (GCP). Each provider offers a comprehensive suite of services, catering to different customer needs. AWS, as the market leader, provides an extensive range of services and global infrastructure. Microsoft Azure combines a broad set of cloud services with seamless integration with existing Microsoft technologies, making it a popular choice for organizations. GCP, known for its expertise in data analytics and machine learning, offers advanced cloud services to enable cutting-edge applications.

Comparing the major cloud providers involves assessing their strengths and weaknesses. AWS is good in terms of **service variety**, **scalability**, and **maturity**, but it may present a **steeper learning curve for beginners** and require thorough architectural planning. Microsoft Azure, the Microsoft top-class product for Cloud Computing, provides **seamless integration with Microsoft technologies**, offering a **familiar environment for enterprises**, along with a vast array of services. Google Cloud Platform (GCP) stands out in **data analytics**, **machine learning**, and **big data processing**, making it a preferred choice for organizations focusing on AI-driven applications. However, GCP's service portfolio may not be as extensive as AWS or Azure.

Looking to the future, cloud computing is expected to witness further advancements and trends. Edge computing is gaining prominence, enabling processing and storage to occur closer to the data source, reducing latency and improving real-time decision-making. Serverless computing, which abstracts infrastructure management

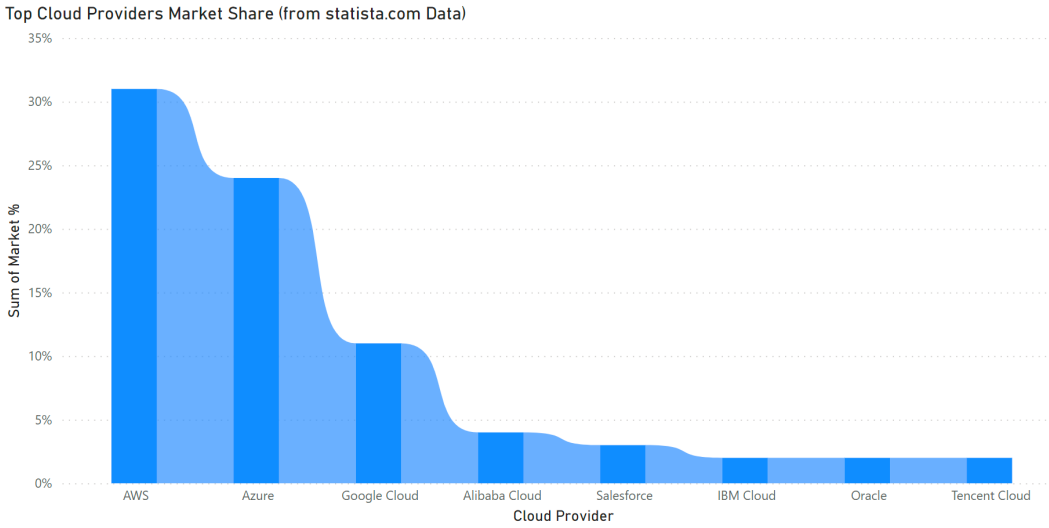


Figure 2: Cloud Computing Providers Market Share
Source: Data taken from `https://statista.com`

and allows developers to focus solely on code, is becoming increasingly popular, offering enhanced scalability and cost efficiency. Additionally, the integration of AI and ML technologies into cloud platforms holds immense potential for automating tasks, detecting anomalies, and optimizing resource allocation. [53]

In conclusion, cloud computing has revolutionized the way businesses and individuals leverage computing resources, providing unparalleled scalability, flexibility, and cost-efficiency. Born out of the need to overcome traditional infrastructure limitations, cloud computing has evolved rapidly, with major providers like AWS, Azure, and GCP dominating the market. While each provider has its strengths and weaknesses, organizations can choose the one that best aligns with their specific requirements.

Looking ahead, cloud computing will continue to evolve, with trends like edge computing, serverless computing, and AI integration shaping the future of this transformative technology. There will be more concious usage of Machine Learning applied to Artificial Intelligence, resulting in more services increasingly focused on customers' primary needs, the automation of laborious, manual, and costly processes, and cost reduction.

1.2 Microsoft Azure

As mentioned in the previous introduction, Microsoft's response to the Cloud revolution is represented by Microsoft Azure, whose "official" history begins with the launch of *Windows Azure* in 2010, influenced by previous projects and initiatives. [48] Before the official launch, Microsoft had undertaken various initiatives in the field of Cloud Computing: starting from 2006, Microsoft introduced a series of live services called *Windows Live*, which aimed to provide the first SaaS solutions for customers. Among the various services offered, we can find:

- **Windows Live Mail**: An email client intended to replace Outlook on desktop computers.

- **Windows Live Messenger**: A popular instant messaging client that allowed calling, video calling, chatting, file sharing, and after its acquisition by Skype in 2011, it was integrated into Skype.
- **Windows Live SkyDrive**: The predecessor to OneDrive, offering free online storage space to upload and share files, accessible from any device with an internet connection.
- **Windows Live Writer**: A tool for bloggers that allowed users to publish and write posts on their blogs through advanced formatting and a WYSIWYG (what you see is what you get) editor.

As the years have passed, some services have been discontinued, others expanded, and still others integrated into other solutions. Nevertheless, the impact of these services, the precursors to many current solutions, is still evident in many of Microsoft's present-day platforms.

1.3 Compliance and Regulatory Standards

As cloud computing becomes increasingly integral to our digital lives, understanding the role of compliance in this sphere is more important than ever. Compliance in cloud computing is a vital part of ensuring data security, maintaining privacy, and building a foundation of trust between cloud service providers (CSPs) and their clients. Microsoft Azure is the leader in addressing these compliance challenges, offering a robust framework to ensure that its services not only meet but often exceed the stringent requirements of regulatory standards worldwide. As the second cloud service provider, Azure is **committed** to maintaining a high level of security and compliance, thereby enabling organizations across various industries to leverage cloud technology while ensuring the protection of sensitive data and adherence to legal and regulatory obligations.

Azure's compliance framework is built around a comprehensive set of *policies*, *controls*, and *procedures*. This framework is designed to comply with a wide array of global and regional standards, including but not limited to *ISO 27001*, the *General Data Protection Regulation* (GDPR), the *Federal Risk and Authorization Management Program* (FedRAMP), the **California Consumer Privacy Act** (CCPA). Through rigorous third-party audits and certifications, Azure demonstrates its commitment to upholding the highest standards of compliance and data protection. [69]

Azure's global presence need to meet a diverse set of compliance standards, tailored not only to global regulations but also to local laws and industry-specific requirements. For instance, in the healthcare sector, Azure ensures *HIPAA compliance* for its services, providing secure and compliant cloud solutions for **healthcare providers and insurers**. Similarly, for **financial services**, Azure meets the requirements of the *Payment Card Industry Data Security Standard (PCI DSS)*, enabling financial institutions to securely process and store payment information in the cloud.

To assist organizations in managing their compliance posture, Azure offers a suite of tools and features. Azure Policy helps enforce organizational standards and assess compliance at scale, automatically implementing remediation actions for identified issues. The **Microsoft Defender For Cloud** provides unified security

management and advanced threat protection across hybrid cloud workloads, offering insights into the security posture of resources and recommending actions to improve it [49]. Additionally, the **Azure Compliance Manager** simplifies compliance processes by providing a centralized dashboard to manage compliance activities, assess risks, and follow detailed guidance to meet regulatory requirements [27].



Figure 3: Microsoft Defender for Cloud
Source: `https://azure.microsoft.com/en-us/products/defender-for-cloud/`

Azure empowers organizations to confidently embrace cloud computing by providing comprehensive compliance documentation, best practices, and detailed reports. By leveraging Azure's compliance capabilities, organizations can streamline their compliance efforts, reduce complexities, and focus on their core business objectives. Moreover, Azure's continuous investment in compliance ensures that its services remain at the cutting edge of regulatory standards, offering peace of mind to organizations concerned about evolving compliance landscapes.

In an era where digital transformation is key to organizational success, the importance of compliance in cloud computing cannot be overstated. Microsoft Azure's commitment to compliance and data protection positions it as a trusted partner for organizations navigating the complexities of regulatory requirements. By offering a secure, compliant, and feature-rich cloud platform, Azure enables businesses to achieve their digital transformation goals while maintaining compliance and fostering trust among their stakeholders.

1.4 Data Sovereignty and Localization

Data sovereignty refers to the concept that digital data is subject to the laws and governance structures within the nation it is stored. As businesses increasingly move

their operations to the cloud, understanding and complying with data sovereignty laws have become paramount. These laws dictate how data is handled, stored, and transferred across borders, impacting decisions on data management and cloud service selection.

The importance of data sovereignty extends beyond legal compliance. It encompasses issues of privacy, security, and geopolitical considerations. As nations become more protective of their data, organizations must navigate a complex landscape of regulations to ensure their cloud deployments do not inadvertently violate local laws.

Microsoft Azure addresses data sovereignty concerns through a strategic global network of data centers, enabling organizations to store data in specific regions in compliance with local regulations. Azure's infrastructure spans multiple countries, offering localized services that adhere to national laws and standards. This localization ensures that customers can manage their data residency, sovereignty, and compliance requirements effectively. Azure makes it a priority to be transparent about where customer data is stored, processed, and managed. The Azure Trust Center provides detailed information on Azure's global infrastructure, including the location of data centers and the compliance certifications they meet. This transparency allows organizations to make informed decisions about their cloud deployments, aligning with data sovereignty requirements.

To further support data sovereignty, Azure offers a range of services that enable fine-grained control over data. These include:

- **Azure Policy**: Helps enforce organizational standards and assess compliance with external regulations, allowing for automated governance and ensuring data is handled according to sovereignty requirements.
- **Microsoft Defender For Cloud**: Offers advanced threat protection and security management for services hosted in Azure, ensuring data is protected in accordance with local laws.
- **Geo-Redundant Storage** (GRS): Enables data replication in multiple locations within a single region or across two regions, providing flexibility in managing data residency while ensuring data durability and high availability.
- **Microsoft Cloud for Sovereignty**: a platform introduced in 2022 that creates software boundaries within the cloud to provide the additional protection that government entities require, utilizing hardware-based confidentiality and encryption controls.

Despite Azure's comprehensive approach to data sovereignty, organizations may face challenges due to varying and sometimes conflicting regulations across jurisdictions. To navigate these challenges, Azure continually updates its services and compliance offerings, working closely with legal and regulatory bodies worldwide. Azure also provides extensive documentation and best practices to help organizations understand and meet their data sovereignty obligations.

1.4.1 Microsoft Cloud for Sovereignty

This service is designed to help governments embrace digital innovation while ensuring digital sovereignty, overcoming the challenges of maintaining digital sovereignty in a public cloud environment. It allows for the modernization of government services without compromising on security, offering a pathway to digitize manual processes

and enhance user experiences for citizens and government employees alike. Microsoft Cloud for Sovereignty aims to support social and economic growth through the right technology platforms. [7]

Its core include:

- Utilizing Azure public cloud capabilities as its foundation.
- Offering regulatory compliance and transparency into cloud operator activities.
- Providing sovereign guardrails through codified architecture, workload templates, localized Azure Policy Initiatives, tooling, and guidance.
- Advanced sovereign control services like Azure Confidential Computing and Azure Key Vault Managed HSM to prevent operator access to data and resources, enhancing data sovereignty.

Azure's robust global infrastructure and localized services empower businesses to meet their data sovereignty requirements while leveraging the scalability, flexibility, and efficiency of cloud computing. By prioritizing compliance and offering tools for effective data management, Azure ensures that organizations can navigate the complex landscape of data sovereignty with confidence.

1.5 Azure Governance and Cost Management

In the evolving cloud ecosystem, governance is pivotal to ensure that cloud resources are utilized effectively, securely, and in alignment with organizational policies and compliance requirements. **Azure governance** encompasses the strategies, processes, and technologies that enable organizations to manage, monitor, and enforce their rules and policies on their Azure resources.

Azure provides a comprehensive suite of tools and features designed to facilitate robust governance and effective management of cloud resources. These tools help organizations maintain control over their cloud environment, ensuring that their Azure deployments remain compliant, secure, and optimized for cost.

- **Azure Policy**: A key component of Azure's governance framework, Azure Policy, allows organizations to create, assign, and manage policies that enforce rules over their Azure resources. It helps ensure compliance with internal guidelines and external regulations by automatically checking resource configurations and taking corrective actions [16].
- **Azure Blueprints**: Azure Blueprints simplifies the deployment of repeatable cloud environments by defining a package of resources that adhere to an organization's standards, patterns, and requirements. It helps streamline the setup of governed environments, ensuring consistency across deployments [14].
- **Azure Management Groups**: These provide a level of scope above subscriptions, allowing organizations to efficiently manage access, policies, and compliance across multiple Azure subscriptions. Management groups simplify the hierarchy of resources, making it easier to apply governance controls at scale [15].

- **Azure Cost Management and Billing**: This tool provides a comprehensive set of features to monitor, allocate, and optimize cloud spend. Azure Cost Management and Billing offers insights into your spending patterns, identifies underutilized resources, and provides recommendations for cost savings, enabling organizations to get the most value from their Azure investments [28].

Implementing effective governance and cost management strategies in Azure [6] requires a comprehensive approach that includes policy enforcement, resource organization, and continuous cost optimization.

- **Implement Policies for Compliance and Security**: Use Azure Policy to define and enforce compliance standards and security practices. Regularly review and update policies to align with evolving compliance requirements and security threats.
- **Organize Resources for Visibility and Control**: Utilize Azure Management Groups, subscriptions, and resource groups to organize cloud resources logically. This organization supports better visibility and granular control over resources, facilitating easier governance.
- **Optimize Costs Through Azure Cost Management**: Regularly monitor cloud spend using Azure Cost Management and Billing. Identify and eliminate waste by resizing or shutting down underutilized resources, and take advantage of Azure's pricing options, such as reserved instances for predictable workloads.
- **Leverage Azure Blueprints for Standardized Deployments**: Use Azure Blueprints to deploy pre-configured environments that meet your organizational standards. This ensures consistency and compliance across all Azure deployments

By leveraging Azure's governance and cost management tools, businesses can achieve a delicate balance between innovation and control, driving forward their digital transformation initiatives while keeping a vigilant eye on compliance, security, and budgetary constraints. This strategic approach to cloud management is essential for any organization looking to thrive in today's fast-paced digital landscape.

Chapter 2

Evolution and Diversity of Database Systems

The journey of **Database Management Systems** (DBMS) is a fascinating story of evolution, driven by the growing needs of businesses to store, retrieve, and analyze data. From the early days of flat-file systems to the sophisticated databases we use today, the development of database technologies reflects the changing landscape of computing and information management. [67]

2.1 Historical Introduction to Databases

Data has invariably been a critical asset for organizations globally, a fact that is increasingly significant in the contemporary era. The capability to effectively store, manage, and analyze data is essential for informed decision-making, strategic planning, and operational efficiency, particularly in an age significantly influenced by artificial intelligence. This requirement has prompted the development of sophisticated systems capable of efficiently accommodating substantial volumes of data—databases. These systems are distinctly varied, designed to address diverse specifications and needs.

The story of the database systems begins in the early days of computing, where the need for the first organized data storage emerged. It is a story that reflects the evolution of technologies, the increasing complexity of data management, the growing demands of businesses and innovation. We can identify the start in what is called "pre-digital" era, where data was manually sorted and filed in physical storage systems. With the advent of computers, the first electronic databases emerged. Initially they were simple flat-file systems where data was stored in plain text files.

The real revolution in databases came in 1970 with the brilliant intuition of **Edgar F. Codd**, that presented *A Relational Model of Data for Large Shared Data Banks* [8]. Codd, which was an IBM researcher at that time, proposed a system where data could be stored in tabled, with rows representing records and columns representing attributes. For the first time, a model was introducing concepts like data independence where the physical storage of data could have been abstracted from its logical structure. This innovation formed the solid groundwork for the development of **SQL** (Structured Query Language), the standard de facto for managing relational databases [56].

Following this revolution, the 1980s and 1990s saw the rise of **RDBMS**, Relational Database Management Systems, with Oracle, IBM, DB2 and Microsoft SQL Server leading the business. These systems implemented the relational model and made it accessible to the organizations of all sizes. The RDBMS era, moreover, was marked with a focus on **ACID** (Atomicity, Consistency, Isolation, Durability) properties [63], ensuring reliable transaction processing and data integrity.

With the internet revolution started in 2000s, companies faced new challenges in data management, with a particular concern in scale and flexibility. Big data and real-time applications demanded too much from traditional RDBMS, and this led to the emergence of **NoSQL databases**, which provided more flexible schema and were designed to scale horizontally across many servers. These servers became popular for applications requiring high-scalability and the ability to handle various types of data.

Both RDBMS and NoSQL are great tools, but with some limitations. This led to the necessity of a new system, known with the name **NewSQL** [1], aiming to combine the scalability and performance of NoSQL with the strong consistency and transaction support of RDBMS. In parallel, advancements in distributed systems, cloud computing, and in-memory databases have further diversified the database landscape, offering solutions for virtually every data storage and processing need.

Today, the database ecosystem is more diverse than ever. Cloud-based databases and Database as a Service (DBaaS) offerings are making powerful data management capabilities accessible to even the smallest organizations. Artificial intelligence and machine learning are being integrated into database systems, enabling more advanced data analysis and automation. Meanwhile, the ongoing development of open-source databases continues to democratize access to cutting-edge data management technologies. As we look to the future, the evolution of database technology promises to be as dynamic and in evolution as its past.

2.2 Diversity of Database Models

2.2.1 Relational Database Systems

At the heart of every RDBMS is the relational model [10], which organizes data into tables (or relations) consisting of rows and columns. Each table represents a different entity type, and each row within a table represents a single instance of that entity. Columns represent attributes of the entity, and the unique key identifies each row. This structure allows for data to be easily accessed, related, and manipulated using Structured Query Language (SQL), a powerful and standardized language specifically designed for managing and querying data in a relational database. Its key point stands in its simplicity and the structured way it stores data, allowing for powerful querying and data manipulation. SQL language is needed to query and manage data and perform some other operations like users management. These database types are known for their reliability and the consistency they offer, particularly in terms of transaction processing. Despite the variety of database type that were developed during the years, the relational databases remain prevalent in many

applications and scenarios because of their solid and mature technology, extensive support and familiarity of SQL among developers and administrators.

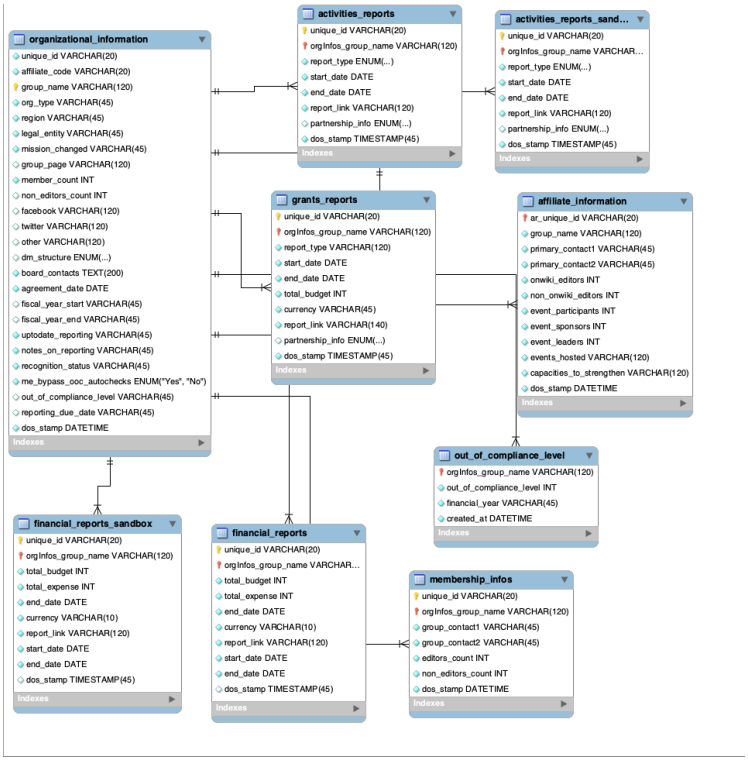


Figure 4: Relational Database Schema Example
Source: `https://wikimedia.org`

Core features of RDBMS [59] can be summarized to:

- **ACID Compliance**: RDBMS are known for their ACID (Atomicity, Consistency, Isolation, Durability) properties, ensuring reliable transaction processing and data integrity. This makes RDBMS ideal for applications requiring complex transactions, such as financial systems.
- **Data Integrity and Security**: relational databases provide robust data integrity features, including primary and foreign keys, data types, and constraints that enforce the accuracy and consistency of data. They also offer comprehensive security measures, such as authentication, authorization, and encryption, to protect sensitive information.
- **Scalability and Performance**: modern RDBMS are designed to scale vertically (by adding more powerful hardware) and, more recently, horizontally (through techniques like sharding) to manage larger volumes of data and user load, ensuring high performance and availability.
- **Data Modeling and Relationship Representation**: the relational model excels at representing data relationships through foreign keys and join operations, enabling complex data analysis and reporting.

Providing lots of advantages and benefits, like:

- **Flexibility and Ease of Use**: the use of SQL for data manipulation and the tabular representation of data make RDBMS particularly user-friendly and adaptable to a wide range of applications.
- **Data Accuracy**: by enforcing data integrity constraints, RDBMS ensure the accuracy and consistency of stored data.
- **Mature Ecosystem**: decades of development have led to a mature ecosystem of tools, best practices, and community support around RDBMS, making them a reliable choice for critical data storage needs.

Despite their advantages, RDBMS face challenges in the era of big data and cloud computing:

- **Handling Unstructured Data**: the structured nature of relational databases can make it difficult to store and process unstructured data, such as text, images, and video, which is where NoSQL databases often have an edge.
- **Scalability Limits**: while modern RDBMS have made strides in scalability, the sheer volume and velocity of data generated today can strain traditional relational databases, pushing organizations to explore distributed databases or NoSQL solutions for certain use cases.
- **Complexity and Overhead**: the relational model's complexity and the need for schema designs upfront can introduce overhead in application development, especially for projects with rapidly evolving data models.

RDBMS have been a critical element in the field of data management for decades, offering a reliable, secure, and efficient means of storing and querying structured data. While the rise of big data and the proliferation of NoSQL technologies have introduced new paradigms in data storage and processing, RDBMS continue to evolve, integrating new features to address modern challenges. Their unmatched ability to ensure data integrity, support complex transactions, and facilitate deep relational data analysis ensures that RDBMS will remain a critical component of the data management landscape for years to come.

2.2.2 NoSQL Databases

NoSQL databases represent a broad category of database management systems that differ from traditional relational database management systems (RDBMS) in several key ways. They are designed to handle a wide variety of data models, including **key-value**, **document**, **wide-column**, and **graph formats**. These databases are particularly well-suited for managing large volumes of structured, semi-structured, and unstructured data, and are known for their scalability, flexibility, and high performance.

There are different types of NoSQL databases, in particular:

- **Key-Value Stores**: these are the simplest form of NoSQL databases, storing data as a collection of key-value pairs. They are highly partitionable and allow for horizontal scaling. Examples include **Redis** and **Amazon DynamoDB**.
- **Document Stores**: these databases store data as JSON, BSON, or XML documents. They are flexible and intuitive for developers to use, making them

popular for web application development. **MongoDB** and **Couchbase** are notable examples.

- **Wide-Column Stores**: these databases store data in tables with rows and columns, but unlike RDBMS, the name and format of columns can vary from row to row in the same table. This type is optimized for queries over large datasets and column-based storage. **Cassandra** and **Google Bigtable** are key examples.
- **Graph Databases**: designed to store and navigate relationships, graph databases are ideal for data that is interconnected and best represented as a graph. They are commonly used for social networks, fraud detection, and recommendation engines. Examples include **Neo4j** and **Amazon Neptune**.

Providing several advantages:

- **Scalability**: NoSQL databases are designed to expand horizontally, making them well-suited for cloud computing and big data applications.
- **Flexibility**: they allow for the storage of unstructured and semi-structured data, offering flexibility in data models.
- **High Performance**: many NoSQL databases are optimized for specific data models or operations, leading to high performance for particular types of applications.
- **High Availability**: features like replication and sharding are built into NoSQL databases, enhancing data availability and resilience.

NoSQL databases offer a powerful alternative to traditional relational databases, providing scalability, flexibility, and performance for handling big data and real-time web applications.

2.3 Present Time

Today's database landscape is diverse, with organizations selecting systems based on specific use cases, performance requirements, and scalability needs. According to *DB-Engine*'s ranking chart LINK46, at first three places we find, in order, **Oracle**, **MySQL** and **SQL Server**, meaning that relational DBMS rules the world of data. Then the chart continues with *PostgreSQL*, *MongoDB*, *Cassandra*, *DynamoDB* and the italian *Redis*:

- **Oracle Database and Microsoft SQL Server** continue to be leaders in the enterprise RDBMS space, known for their robustness, comprehensive features, and support for complex transactions.
- **MySQL and PostgreSQL** are popular open-source RDBMS choices, widely adopted for web applications due to their performance and reliability.
- **MongoDB** has become synonymous with document-oriented databases, praised for its flexibility in handling semi-structured data and its ease of use in development.

- **Cassandra and DynamoDB** offer scalable solutions for applications requiring distributed database systems, capable of handling large volumes of data across many commodity servers.
- **Redis** is widely used for caching and real-time analytics, offering exceptional performance for high-speed transactions.

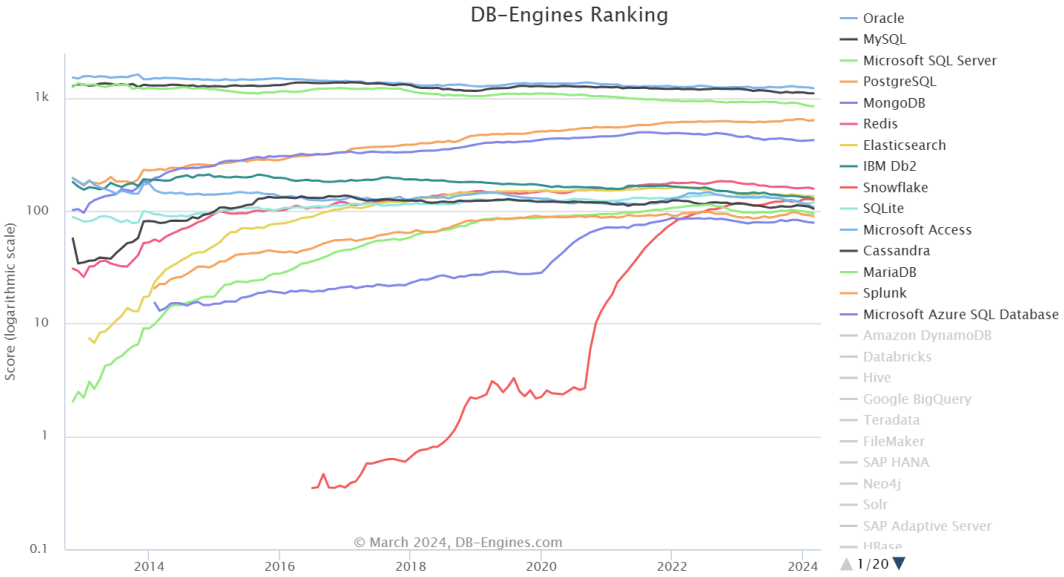


Figure 5: DB-Engine's DB Ranking Chart

Source: `https://www.db-engines.com`

The current database landscape offers a rich selection of options catering to various needs and challenges of modern applications. From the steadfast reliability and consistency of RDBMS to the flexibility and scalability of NoSQL databases, and the innovative approach of NewSQL, developers and organizations have the tools to build efficient, scalable, and robust systems. As data continues to drive decision-making and innovation, choosing the right database solution becomes a pivotal aspect of technology strategy, impacting application performance, scalability, and ultimately, business success.

2.4 Going ahead: NewSQL

The evolution of database technology continues to be a critical area of development, shaping how data is stored, accessed, and utilized across various applications and industries. The advent of **NewSQL databases** has sparked significant interest, proposing solutions that combine the best of both traditional relational databases (RDBMS) and NoSQL systems. NewSQL databases emerged as a response to the limitations of traditional RDBMS in handling massive scales of data and the demand for more flexible, scalable solutions offered by NoSQL systems **without sacrificing ACID** (Atomicity, Consistency, Isolation, Durability) guarantees and SQL's expressive querying capabilities. NewSQL aims to provide the scalability and distributed processing of NoSQL systems while maintaining the transactional consistency, reliability, and ease of query associated with RDBMS.

2.4.1 NewSQL Databases

- **Google Spanner**: perhaps the most famous example, Spanner is designed to scale horizontally across thousands of servers while providing global consistency and robust transactional support. It introduces the concept of globally synchronized clocks (TrueTime) to manage transactions across distributed systems.
- **CockroachDB**: inspired by Google Spanner, **CockroachDB** is an open-source, distributed SQL database that focuses on scalability, consistency, and survivability, making it ideal for cloud applications that require reliable data storage across multiple geographic locations.
- **TiDB**: TiDB is another notable NewSQL database that offers horizontal scalability, strong consistency, and high availability. It's designed to support hybrid transactional and analytical processing (HTAP), allowing users to perform real-time analytics on transactional data.

The future of database technology is marked by rapid innovation and a shift towards systems that offer the scalability, flexibility, and efficiency demanded by modern applications while not compromising on the transactional integrity and ease of use that have made traditional databases foundational to IT systems. As NewSQL and other emerging technologies continue to evolve, they promise to redefine the landscape of database systems, catering to the ever-expanding scale of data and the complexity of global, data-driven operations. The journey of database evolution is far from complete, and the next generation of databases will undoubtedly provide even more powerful tools for harnessing the full potential of data in the digital era.

Chapter 3

Microsoft Azure for Database Management

Microsoft Azure provides a suite of cloud services, including several options for relational database management systems (RDBMS). The story of relational databases in Azure reflects the broader evolution of cloud computing and database technology, aiming to offer scalable, reliable, and flexible database solutions to meet a wide range of application needs.

3.1 Relational Databases

3.1.1 Azure SQL

Azure SQL Database offers a set of services that give you SQL Server's database features in the cloud, making it compatible with existing SQL Server apps. It makes managing databases easier by handling updates, backups, and patches for you. This lets developers focus on their data and building their applications instead of worrying about database upkeep.

SQL Server first appeared in 1989 [60], and its cloud counterpart was introduced at the Microsoft Professional Developers Conference in 2008 by Ray Ozzie, initially called `Windows Azure`. It was later renamed Microsoft Azure, and SQL Azure became Azure SQL, expanding its services significantly. Azure SQL provides various options for deployment, offering different levels of flexibility:

- **SQL on Virtual Machines**: great for migrations and apps that need OS-level access.
- **Managed Instances**: ideal for most migrations to the cloud.
- **Databases**: perfect for modern cloud apps, with options for hyperscale and serverless.

SQL Server on Virtual Machine runs SQL Server in an Azure VM just like the desktop version, so moving to the cloud is straightforward. Azure assists with automatic backups and security patches, known as *Infrastructure as a Service* (IaaS). This means customers handle updates and patches for SQL Server and the VM's OS, except for critical security patches. You get full product capabilities but are responsible for maintenance. Customers often wonder before choosing this: "I want

to move to the cloud quickly but keep control over the operating system and full SQL Server functionality" [35].

The other two options, Azure SQL Managed Instance and Azure SQL Databases, are *Platform as a Service* (PaaS) solutions. They offer a managed database engine that handles most management tasks like upgrades, patching, and backups, including many features and functions:

- **Business continuity**: keeps your business running smoothly, even during unexpected events.
- **High availability**: ensures databases are accessible 99.99% of the time, with no worries about maintenance or outages.
- **Automated backups**: use Azure's geo-redundant storage for backup copies, adding an extra layer of safety.
- **Long-term backup retention**: lets you keep selected databases safe for up to 10 years.
- **Geo-replication**: creates copies of your database in the same or different datacenters.
- **Scalability**: easily increases resources like CPU, memory, and storage without waiting.
- **Network security**: includes firewalls, Azure Private Link, and integration with virtual networks to keep your data safe.
- **Advanced security**: finds and stops threats to your databases, keeping your data secure.
- **Automatic tuning**: looks at your workload and suggests ways to improve your app's performance.
- **Built-in monitoring**: helps you understand how your databases are doing and find ways to fix any issues.
- **Built-in intelligence**: spots potential problems in your workload and offers advice on fixing them.

A key difference between IaaS and PaaS is that SQL Database and SQL Managed Instance don't tie you to a specific SQL Server version. They're versionless, with Azure updating and fixing things as needed.

SQL Managed Instance is a PaaS option for Azure SQL. It gives you a SQL Server instance without much of the management hassle. Most SQL Server features are here, ideal for those who want instance-specific features without reworking their apps. Instance-specific features include things like SQL Server Agent and Service Broker, among others. With SQL Managed Instance, you use these features without managing or accessing the underlying OS or infrastructure. Before choosing this, customers often ask: "I want to move to the cloud and cut down on management tasks, but I still need instance-specific features."

SQL Database is another PaaS option of Azure SQL that takes away the need to manage both the OS and SQL Server instance. It's all about giving you a database to work with right away, perfect for building modern applications. It's unique because it supports unlimited database storage (hyperscale) and can automatically adjust for varying workloads (serverless), offering the highest availability SLA. Microsoft's management means it also brings smart monitoring and performance features. Customers typically consider this option when they say: "I want to create modern apps, possibly for many users, with the best uptime and reliable performance."

3.1.2 Azure Database for MySQL, PostgreSQL, and MariaDB

Azure Database for MySQL, PostgreSQL, and MariaDB are fully managed relational database services that bring the power, community support, and familiarity of these popular open-source databases to the cloud. Microsoft Azure provides these services with built-in high availability, security, and scalability, making it easier for developers to deploy, manage, and scale their database solutions without the overhead of managing infrastructure.

The introduction of these services by Microsoft Azure is part of a broader strategy to embrace open-source technologies and provide a comprehensive set of cloud-based services that cater to various databases and application needs. Azure Database for PostgreSQL and MySQL were both announced in May 2017 [11], while Azure Database for MariaDB was later introduced in 2018 [45]. These services were developed to offer developers and enterprises the flexibility of their favorite open-source databases with the benefits of a managed cloud service.

- **Azure Database for MySQL**: offers a managed MySQL database service, providing full compatibility with MySQL Community Edition. It allows developers to leverage MySQL's features and tooling in a fully managed Azure environment.
- **Azure Database for PostgreSQL**: delivers a managed PostgreSQL database service, supporting the latest PostgreSQL version. This service integrates seamlessly with existing PostgreSQL ecosystems.
- **Azure Database for MariaDB**: provides a managed MariaDB database service, ensuring full compatibility with MariaDB community editions. It's ideal for those already using MariaDB and looking to migrate or scale their databases in the cloud. It will be retired in 2025, as part of Microsoft strategy to invest in peculiar fields of open-source databases [45].

The key charateristics and features of these services include:

- **Fully Managed**: these services are fully managed, meaning Azure handles routine tasks such as provisioning, backups, patching, monitoring, scaling, and security.
- **Built-in High Availability**: they offer built-in high availability with a 99.99% uptime SLA, ensuring that database applications are always up and running.

- **Security**: Azure provides advanced security features, including network isolation using Azure Virtual Network, data encryption at rest and in transit, and integrated authentication and authorization capabilities.
- **Scalability**: users can scale compute and storage resources with minimal downtime, allowing applications to efficiently manage fluctuating workloads.
- **Compliance**: these services comply with a broad set of international and industry-specific standards, including GDPR, HIPAA, ISO, and more, ensuring that data is protected and managed according to regulatory requirements.

3.2 Non-Relational Databases

3.2.1 Cosmos DB

Azure Cosmos DB is Microsoft's globally distributed, multi-model database service, designed to enable high-performance, scalable applications across the globe. Offering turnkey global distribution, seamless scalability, and support for multiple data models within the same database engine, Cosmos DB represents a significant evolution in the landscape of database solutions. Cosmos DB was born in 2010 with the internal name of *Project Florence*, aiming to address the needs of large-scale social media applications within Microsoft, like Xbox Live [2]. It was designed to provide global distribution and horizontal scalability from the ground up, which were challenging to achieve with traditional relational database systems. Officially launched as *Azure DocumentDB* in 2014, it initially offered document-oriented storage. In 2017, it was renamed to **Azure Cosmos DB**, expanding its capabilities to support multiple data models and APIs, including **MongoDB**, **Apache Cassandra**, **Gremlin (for graph)**, and **Azure Table Storage**, in addition to its original **SQL (Core) API for document data**.
Cosmos DB is a fully managed NoSQL database service. It is designed to operate at a massive scale globally, with built-in support for distributing data across any number of Azure's geographic regions to provide low-latency access to data, regardless of where users are located. It supports schema-less data, which makes it suitable for a wide range of applications, from IoT to mobile apps and everything in between.

Cosmos DB differs from traditional relational databases in several key ways:

- **Data Model Flexibility**: while relational databases use a structured schema and store data in tables, Cosmos DB supports multiple data models and is schema-agnostic. This flexibility allows for rapid development and iteration, accommodating the diverse structures of data generated by modern web and mobile applications.
- **Global Distribution**: relational databases typically require significant effort to distribute data globally, including complex replication and sharding strategies. Cosmos DB, however, is designed from the ground up for global distribution, making it easy to distribute and access data worldwide.
- **Scalability**: scaling a relational database often involves vertical scaling (adding more power to the existing machine) or manual sharding (partitioning data across multiple databases). Cosmos DB, on the other hand, offers seamless

horizontal scaling (adding more machines) across global regions, without the need for manual intervention.

- **Consistency Models**: relational databases traditionally offer strong consistency. Cosmos DB provides multiple consistency models, from strong to eventual, allowing developers to make trade-offs between consistency, availability, and latency to best meet their application's requirements.

Cosmos DB supports multiple APIs that determine how you interact with data and which data models you can use. These APIs allow Cosmos DB to support different types of applications seamlessly. The deployment choice can be made based on the specific requirements of your application, such as the need for document, graph, key-value, or column-family data models. The available APIs are:

- **SQL (Core) API**: provides a rich querying language that is highly compatible with JSON data, making it ideal for document model applications.
- **MongoDB API**: offers compatibility with MongoDB, allowing existing MongoDB applications to migrate to Cosmos DB without changing the application code.
- **Cassandra API**: enables applications that are built for Apache Cassandra to migrate to Azure Cosmos DB with minimal changes.
- **Gremlin API**: supports graph-based data models, making it suitable for applications that deal with connected data, such as social networks or recommendation engines.
- **Table API**: designed for applications requiring key-value storage, providing seamless integration with existing Azure Table storage applications.

3.2.2 Azure Cache for Redis

Azure Cache for Redis is Microsoft's secure, highly available, and performant caching service built on the popular open-source Redis cache. It enhances the performance of applications by providing high-speed access to data, significantly reducing the latency and increasing the throughput of data-driven applications. Redis (**Remote Dictionary Server**), started as an open-source project in 2009 by Salvatore Sanfilippo [70] [4]. It quickly became popular due to its simplicity, speed, and versatility in storing key-value data. Azure Cache for Redis brings this powerful technology to the Azure ecosystem, offering it as a managed service that integrates seamlessly with other Azure services. Launched in 2013, **Azure Cache for Redis** allows developers to incorporate a highly accessible and low-latency caching layer into their applications, enhancing performance and user experience.

While traditional databases store data on disk or SSDs, Azure Cache for Redis primarily stores its data in memory. This fundamental difference provides Azure Cache for Redis with several advantages like:

- **Speed**: in-memory data access is significantly faster than disk-bound access, making Redis ideal for operations requiring quick read and write operations.

- **Simplicity**: Redis uses a key-value store model, which, while less complex than relational databases, is incredibly flexible, supporting data structures like strings, hashes, lists, sets, and sorted sets.
- **Caching and Session Store**: unlike traditional databases designed for long-term data persistence, Azure Cache for Redis excels at temporary data storage, such as caching and session storage, to enhance application performance.
- **Latency Reduction**: by caching frequently accessed data, applications can reduce the need to fetch data from slower, disk-based databases, decreasing latency.
- **Load Reduction**: serving data from the cache reduces the load on the underlying databases, improving their responsiveness and scalability.
- **Session Management**: it can store session data, ensuring fast access and improving user experience in web applications

Azure Cache for Redis can be used in several ways, including **Application Cache** (storing commonly accessed application data such as web page content or results of database queries), **Distributed Session Store** (to keep user session information across web application instances) and also **Real-time Analytics** (for quick analysis and statistic data).

3.2.3 Azure Synapse Analytics

Azure Synapse Analytics, is a limitless analytics service that brings together enterprise data warehousing and Big Data analytics. It offers a unified environment for preparing, managing, and serving data, combining capabilities from both data lakes and data warehouses. The service was first started in 2016 with the introduction of *Azure SQL Data Warehouse*. This service was designed to provide a fully managed, scalable data warehouse solution in the cloud, allowing businesses to store and analyze large volumes of data. The foundation of SQL Data Warehouse was built on the massively parallel processing (MPP) architecture, enabling it to handle massive datasets efficiently by distributing data and query processing across many nodes. It was renamed in **Azure Synapse Analytics** in late 2019.

Azure Synapse Analytics presents a leap forward in analytics, offering a powerful, flexible platform that bridges the gap between data warehousing and big data analytics. By providing tools for data integration, advanced analytics, and machine learning in a single service, Synapse enables businesses to derive insights and value from their data more efficiently than ever before. Its key features are:

- **Integrated Analytics**: Synapse provides a seamless blend of big data and data warehousing technologies, enabling complex analytics across large volumes of data.
- **On-demand Query Processing**: it allows to query data on-demand using serverless on-demand resources or provisioned resources, catering to both occasional and intensive workloads.
- **Unified Data Integration**: Synapse integrates with various data sources, including relational and non-relational databases, data lakes, and big data systems, facilitating easy data ingestion and management.

- **Advanced Security Features**: offers robust security measures, including data masking, dynamic data masking, and column-level security, ensuring data privacy and compliance.
- **Machine Learning and BI Integration**: integrates with Azure Machine Learning and Power BI, among others, to provide advanced analytics, machine learning model training, and visualization capabilities directly within the service.

3.2.4 Azure Table Storage

Azure Table Storage, a component of the Azure Storage services, offers a NoSQL data store for semi-structured data. It is tailored for the storage of large volumes of non-relational data, providing a solution that is both highly available and capable of massive scalability. This makes it an ideal choice for applications in need of a flexible database schema with the capacity for rapid scaling and efficient data querying. As a service, it facilitates the storage of non-relational structured data (NoSQL data) in the cloud, acting as a key/attribute store with a schemaless design. The schemaless nature of Table Storage allows for easy adaptation of data to suit the evolving needs of applications. It offers fast and cost-effective access to data, typically presenting a more economical option than traditional SQL for similar data volumes. This versatility renders Table Storage an excellent repository for flexible datasets, such as user information for web applications, address books, device data, or any other metadata required by a service. A table can house an unlimited number of entities, and a storage account may include any number of tables, subject to the account's storage capacity limit.

Azure Table Storage is managed through the **Azure Portal** [46], **Azure CLI** [29], or **Azure Storage SDKs** [30]. Tables and data can be managed programmatically or through tools like Azure Storage Explorer, which provides a graphical interface for managing Azure Storage resources.

Its key features are:

- **NoSQL Data Model**: Table Storage uses a schema-less design, which means that each row in a table can have a different structure with a different set of fields. This flexibility is ideal for applications that require a flexible data schema.
- **Scalability**: It's designed to scale easily to handle massive amounts of data and traffic, making it suitable for applications with varying performance and scalability requirements.
- **Global Distribution**: when used in conjunction with Azure Cosmos DB's Table API, data can be globally distributed to provide low-latency access to users no matter where they are located.
- **Cost-Effective**: Azure Table Storage offers a cost-effective solution for storing large quantities of data, with a pricing model that charges based on storage used and the amount of data transferred.
- **Security**: it provides secure access to the data using Shared Access Signatures (SAS) and supports encrypting data at rest.

Table Storage is made by the following key components:

- **URL format**: The service employs a specific format for accessing Azure Table Storage accounts: `https://<storage account>.table.core.windows.net/<table>`. Azure tables can be directly accessed using the OData protocol.
- **Accounts**: Access to Azure Storage is mediated through a storage account.
- **Table**: Defined as a collection of entities, tables do not impose a schema on entities, allowing a single table to store entities with different sets of properties.
- **Entity**: An entity, analogous to a database row, comprises a set of properties. In Azure Storage, an entity can be as large as 1MB, whereas in Azure Cosmos DB, an entity's size limit extends to 2MB.
- **Properties**: Properties are name-value pairs within an entity. Up to 252 properties can be used to store data in each entity, alongside three system properties that define a partition key, a row key, and a timestamp. Entities sharing the same partition key can be queried more efficiently and are subject to atomic operations during insertions and updates. The row key serves as a unique identifier for an entity within a partition.

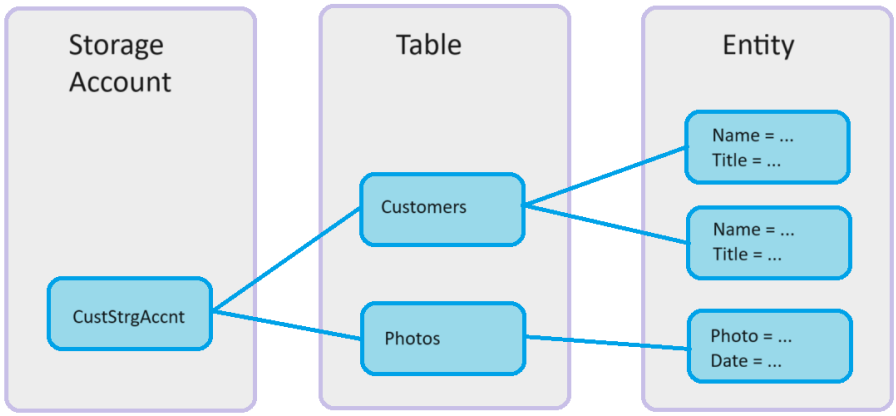


Figure 6: Azure Table Storage components

Chapter 4

Data Security and Privacy Principles

In today's digital era, data acts as a key feature in all organizations and across global economy, and the importance of data security and privacy cannot be underestimated. As businesses and organizations increasingly migrate their operations to the cloud, ensuring the confidentiality, integrity, and availability of data within cloud environments like Microsoft Azure becomes an essential ally.

The migration of databases to cloud platforms such as Azure represents a significant shift in how data is stored, accessed, and managed. While this shift offers unparalleled scalability, flexibility, and efficiency, it also introduces new challenges and complexities in data security and privacy. The shared responsibility model of cloud computing necessitates that both cloud service providers and their customers play active roles in protecting data. Azure provides a robust infrastructure and a suite of tools designed to secure data at rest and in transit. However, customers must also implement best practices in database configuration, access control, and data management to ensure comprehensive protection.

4.1 Foundational Principles of Data Security

Understanding and applying the foundational principles of data security is essential for safeguarding information assets. These principles form the bedrock upon which organizations can build a robust data security strategy, especially when leveraging cloud services like Microsoft Azure. Starting from late 1988 [55] the basis of data security were catelogued into three cornerstone principles: Confidentiality, Integrity, and Availability, collectively known as the **CIA triad**. These principles are still relevant today, and formed the basis of each cloud service, and in particular within Azure databases [31].

4.1.1 Confidentiality

Confidentiality involves ensuring that **sensitive data is accessible only to those who have the authority to view it**. This principle prevents unauthorized access to data, protecting personal and business information from breaches and exposure. In the context of Azure databases, confidentiality is maintained through several mechanisms:

- **Encryption**: Azure provides encryption capabilities for data at rest and in transit. Azure SQL Database, for example, uses **Transparent Data Encryption** (TDE) to encrypt data at rest, ensuring that stored data is unreadable to unauthorized users.
- **Access Controls**: implementing strict access controls through **EntraID** (formerly Azure Active Directory) and **role-based access control** (RBAC) ensures that only authenticated and authorized users can access database resources.
- **Network Security**: Azure's network security features, such as Virtual Network (VNet) service endpoints and firewall rules, restrict access to database resources from unauthorized networks.

4.1.2 Integrity

Integrity refers to the **trustworthiness and accuracy of data throughout its life cycle**. This principle ensures that data is only modified in expected ways and by authorized users, preventing unauthorized alterations that could compromise the data reliability. Azure databases support data integrity through:

- **Data Auditing**: Azure provides auditing features that track database activities, enabling administrators to verify and review changes to data and schema, ensuring that all modifications are authorized and intended.
- **Version Control and Backup**: Azure's support for data versioning and comprehensive backup capabilities allow for the restoration of data to a previous state if unauthorized or accidental modifications occur.

4.1.3 Availability

Availability ensures that **data and database services are accessible to authorized users** when needed, supporting the continuous operation of business processes. Azure enhances the availability of databases through:

- **High Availability Configurations**: Azure databases are designed for high availability with built-in redundancy and failover capabilities. For instance, Azure SQL Database offers automatic failover groups to ensure minimal downtime during planned and unplanned outages.
- **Scalability Features**: Azure's scalability features, such as auto-scaling and elastic pools, ensure that database resources can be dynamically adjusted to meet demand, maintaining performance and accessibility even under heavy load.
- **Disaster Recovery**: Azure's disaster recovery options, including geo-replication and Azure Site Recovery, ensure that data can be quickly restored and services resumed in the event of a disaster.

4.2 Privacy Principles and Compliance

A fundamental objective of Microsoft Azure is to uphold privacy and compliance with regulatory standards while safeguarding data from unauthorized access or

breaches. Privacy principles play a crucial role in guiding organizations towards responsible data management, underscoring the importance of honoring user rights and legal responsibilities. For cloud providers such as Microsoft Azure, it is critical to ensure adherence to essential privacy principles, including Data Minimization, Purpose Limitation, and Consent, and to uphold compliance with regulatory frameworks like the GDPR and CCPA in data storage and processing activities.

4.2.1 Data Minimization

Data Minimization refers to the principle of collecting only the data that is directly necessary to accomplish a specified purpose. This approach limits the amount of personal information that could potentially be exposed in the event of a data breach. In Azure, data minimization [32] can be facilitated through:

- **Designing databases with minimization in mind**: structuring database schemas to only include essential fields.
- **Implementing data purging routines**: regularly deleting or anonymizing data that is no longer needed for the stated purpose.

4.2.2 Purpose Limitation

Purpose Limitation mandates that data collected for one purpose should not be re-purposed without further consent. This principle ensures that individuals' data is not used in unexpected ways, maintaining trust between users and organizations. Azure supports purpose limitation by:

- **Enforcing role-based access control (RBAC)**: ensuring that data access is granted based on the necessity to know for the original purpose of data collection.
- **Audit logs and tracking**: Azure provides comprehensive logging, allowing organizations to monitor and verify that data usage aligns with its intended purpose.

4.2.3 Consent

Obtaining explicit consent from individuals before collecting, processing, or sharing their data is a cornerstone of modern privacy practices. Consent must be informed, specific, and freely given, reflecting a genuine choice by the individual. Azure aids in managing consent through:

- **User authentication and authorization mechanisms**: ensuring that consent decisions are securely recorded and attributed to the correct individual.
- **Privacy policy management tools**: facilitating the clear communication of how data will be used, enabling users to make informed consent decisions.

4.2.4 Compliance Frameworks

Navigating the landscape of privacy laws and regulations, such as the General Data Protection Regulation (GDPR) in the European Union and the California Consumer Privacy Act (CCPA) in the United States, is essential for organizations operating

in or serving customers from those regions. These frameworks set stringent requirements for data protection, rights to access, and the deletion of personal information.

- **GDPR**: emphasizes strong data protection for EU citizens, mandating strict consent protocols, the right to access, and the right to be forgotten, among others [51].
- **CCPA**: grants California residents the right to know about the personal data collected on them, the right to delete personal information, and the right to opt-out of the sale of their personal data [64].

Azure provides several features and services to help organizations comply with these regulations, including:

- **Azure Policy and Compliance Manager**: helps track compliance status with regulatory requirements and implement necessary controls [33].
- **Data Subject Requests (DSRs) tools**: facilitate the processing of requests from individuals exercising their rights under GDPR and CCPA [34].

4.3 Data Security Challenges in Databases

Databases, as the repositories of sensitive and critical information, face numerous security challenges that must be meticulously managed to protect against data breaches, unauthorized access, and other cyber threats. Microsoft Azure gives developers and IT admins powerful tools to identify security holes, fix issues and mitigate incidents.

4.3.1 Vulnerability Management

Vulnerability management involves a cyclical process of identifying, assessing, and mitigating vulnerabilities within database systems. This proactive approach is crucial for preventing potential exploits that could compromise data security [17].

- **Identifying Vulnerabilities**: regularly scanning databases for known vulnerabilities using automated tools and staying informed about new threats.
- **Assessing Vulnerabilities**: evaluating the potential impact of identified vulnerabilities on database security and prioritizing remediation efforts based on risk.
- **Mitigating Vulnerabilities**: applying patches, updates, and configuration changes to address vulnerabilities and reduce the attack surface of databases.

Azure provides integrated security tools, such as **Microsoft Defender for Cloud** (formerly Microsoft Defender For Cloud) [36], which offers vulnerability assessment capabilities, helping organizations to identify and mitigate potential database vulnerabilities efficiently.

4.3.2 Data Encryption

Data encryption is a foundational security measure that protects data confidentiality by converting data into an unreadable format for unauthorized users, both at rest and in transit. Azure can help with encryption in several ways:

- **At-rest Encryption**: protects data stored in databases by encrypting the physical files on disk. Azure offers **Transparent Data Encryption** (TDE) for SQL Database, Azure SQL Managed Instance, and Azure Synapse Analytics [37] to automate the encryption of data at rest.
- **In-transit Encryption**: secures data as it travels across networks to prevent interception by unauthorized parties. Azure supports **SSL/TLS** encryption for data in transit [18], ensuring that data moving to and from Azure databases is encrypted.

4.3.3 Access Control

Access control mechanisms are essential for ensuring that only authorized users can access or modify data within databases. This involves both authentication (verifying user identities) and authorization (determining user access levels).

- **Authentication**: implementing strong authentication methods, including multi-factor authentication (MFA), to verify the identities of users accessing the database.
- **Authorization**: utilizing role-based access control (RBAC) and least privilege principles to ensure users have only the necessary permissions to perform their roles.

Microsoft EntraID integration with Azure databases enhances security by enabling centralized management of user identities and access controls [38].

4.3.4 Data Masking and Tokenization

Data masking and tokenization are techniques used to obscure specific data within a database, making it inaccessible or unintelligible to unauthorized users, thereby protecting sensitive information.

- **Data Masking**: involves hiding real data with modified content (e.g., replacing sensitive fields with generic values). Azure offers Dynamic Data Masking (DDM) for SQL Database and Azure Synapse Analytics [19], allowing administrators to define masking rules for specific database fields without altering the actual data.
- **Tokenization**: replaces sensitive data with non-sensitive substitutes, known as tokens, which can be mapped back to the original data only through a tokenization system. While Azure does not provide native tokenization services, third-party solutions [47] can be integrated with Azure databases to implement tokenization.

4.4 Implementing Privacy by Design in Azure Deployments

Starting from the early 2000 [68], "**Privacy by Design**" has emerged as a fundamental approach, embedding privacy into the design and operation of IT systems, network infrastructure, and business practices. Organizations can implement Privacy by Design principles in Azure deployments, focusing on Privacy Impact Assessments (PIAs), Data Classification, and the Principle of Least Privilege.

4.4.1 Privacy Impact Assessments (PIAs)

Conducting Privacy Impact Assessments (PIAs) is a proactive process to identify and mitigate privacy risks at the early stages of project development and throughout the system lifecycle. In the context of Azure deployments:

- **Early Assessment**: PIAs should be initiated in the planning phase of any project involving personal data to anticipate potential privacy risks.
- **Regular Updates**: as projects evolve, regularly update PIAs to reflect changes in data handling and processing activities.
- **Mitigation Strategies**: identify measures to mitigate identified risks, including the adoption of Azure features that enhance data privacy and security.

Azure provides tools and documentation to aid in conducting PIAs, helping organizations understand how data is managed within Azure services and how to implement appropriate controls.

4.4.2 Data Classification

Data classification [20] involves categorizing data based on its sensitivity and the impact to privacy or security if disclosed or accessed unauthorizedly. Effective data classification in Azure deployments requires:

- **Identification of Sensitive Data**: use Azure Information Protection (AIP) and other classification tools to automatically discover and classify sensitive data across Azure environments.
- **Classification Levels**: define classification levels (e.g., *public*, *internal*, *confidential*, *highly confidential*) and apply them to data stored in Azure based on its sensitivity.
- **Access Controls**: utilize classification labels to enforce access controls and handling requirements, ensuring that sensitive data is adequately protected. Implementing a robust data classification framework is essential for managing data privacy and adhering to regulatory requirements.

4.4.3 Least Privilege Access

The principle of least privilege (PoLP) [21] ensures that individuals and systems have only the minimum levels of access — or permissions — needed to perform their tasks, reducing the risk of unauthorized access to sensitive information. In Azure deployments, PoLP can be achieved through:

- **Role-Based Access Control (RBAC)**: Define roles with specific permissions and assign them to users based on their job functions, ensuring they have access only to the resources necessary for their roles.
- **Conditional Access Policies**: implement conditional access policies in Microsoft EntraID to control access based on conditions such as user location, device compliance, and risk levels.
- **Regular Audits**: periodically review access rights and permissions across Azure resources to ensure adherence to the PoLP and remove unnecessary permissions. Adopting the PoLP across Azure services and resources significantly enhances the privacy and security of deployments by minimizing the potential for unauthorized access or data breaches.

4.5 Best Practices for Secure Database Deployment on Azure

Ensuring the security of databases deployed on Azure requires adherence to a set of best practices that encompass configuration, monitoring, and data resilience strategies.

4.5.1 Security Baseline Configuration

Establishing a security baseline configuration involves setting and applying Azure security benchmarks and best practices right from the initial setup of database services [22]. This foundational step ensures that databases are secured against known vulnerabilities and are compliant with industry standards.

- **Azure Security Benchmarks**: leverage Azure's security benchmarks, which provide guidelines and best practices across different Azure services, including databases. These benchmarks serve as a comprehensive resource for securing your deployments.
- **Principle of Least Privilege**: ensure that database access is tightly controlled. Apply the principle of least privilege by granting users and applications the minimum permissions they need to perform their tasks.
- **Network Security**: Azure Virtual Network (VNet) and network security groups (NSGs) allow isolation of databases and limit access. Implementing firewalls and restricting access to IP addresses known to be safe can further enhance security.

4.5.2 Regular Auditing and Monitoring

Continuous auditing and monitoring [39] are crucial for detecting unusual activities that could indicate a security threat or breach. Azure provides powerful tools to facilitate these processes, enabling proactive responses to potential security incidents.

- **Azure Monitor**: using **Azure Monitor** to collect, analyze, and act on telemetry data from your Azure databases. It helps in understanding the performance and health of your databases, identifying trends, and detecting anomalies.

- **Microsoft Defender For Cloud**: leverage **Microsoft Defender For Cloud** for a unified security management and advanced threat protection across your Azure deployments. It offers continuous security assessment and actionable recommendations to mitigate risks.

4.5.3 Data Recovery and Backup

A comprehensive data recovery and backup strategy [40] is essential for minimizing data loss in the event of an outage, corruption, or breach. Azure offers robust tools and services designed to ensure data durability and recoverability.

- **Automated Backups**: take advantage of Azure's automated backup capabilities for your databases. Azure SQL Database, for example, supports automated backups that are retained for a configurable period, allowing for point-in-time restores.
- **Geo-Replication**: implement geo-replication to create readable secondary replicas of your data in different geographical locations. This not only enhances data availability but also provides a mechanism for disaster recovery.
- **Azure Site Recovery**: for comprehensive disaster recovery solutions, consider using Azure Site Recovery. It helps orchestrate replication, failover, and recovery of Azure virtual machines and on-premises servers, ensuring minimal downtime.

Chapter 5

Database Threats

In today's digital age, databases are the vaults of valuable information and hence a prime target for cyber threats. As the volume of data collected, processed, and stored by organizations has exploded, so too has the attention of cybercriminals intent on exploiting this wealth of information. Understanding the history of cyber attacks on databases, their implications, and the paramount importance of securing data is crucial for any organization navigating the digital landscape. Some of the most prevalent threats to databases are: SQL Injection, Denial of Service (DoS) Attacks, Insider Threats, and Malware and Ransomware.

The history of cyber attacks on databases dates back to the early days of the internet. Initially, these attacks were primarily acts of digital vandalism, aimed at disrupting services or gaining notoriety among the hacker community. However, as e-commerce and online transactions gained traction, attackers quickly recognized the monetary value of sensitive data such as credit card numbers, personal identification information, and intellectual property stored in databases.

One of the first recorded instances of a significant cyber attack targeting databases occurred in the late 1980s with the advent of the SQL Slammer worm. This attack exploited vulnerabilities in Microsoft SQL Server databases, causing widespread internet outages and substantial financial losses. This incident marked a turning point, highlighting the need for robust database security measures.

Over the years, cyber attacks have evolved in sophistication, leveraging advanced techniques such as SQL injection, ransomware, and insider threats. These attacks have far-reaching implications, not only resulting in direct financial losses but also damaging reputations, eroding customer trust, and, in severe cases, leading to legal repercussions and regulatory fines. The 2013 breach of a major retailer's database, resulting in the theft of millions of customers' credit and debit card information, exemplifies the devastating impact of such attacks.

5.1 SQL Injection

SQL Injection remains [66] one of the most critical threats, exploiting vulnerabilities in a database's software to execute unauthorized SQL commands. This type of cyber attack takes advantage of insufficient input validation in web applications to manipulate SQL queries executed by the database. By inserting malicious SQL

code into forms, URLs, or even via direct input fields that a web application uses to communicate with its database, attackers can gain unauthorized access to, manipulate, or destroy sensitive data.

SQL injection attacks are primarily targeted at web applications that use SQL databases. These attacks are possible because many applications fail to properly sanitize user inputs, allowing attackers to inject malicious SQL code. This code is then executed by the database, leading to potential unauthorized access or data manipulation.

The process typically involves an attacker identifying a vulnerable input field in a web application and inserting malicious SQL code into that field. When the application processes the user input without proper validation or sanitization, the malicious code is executed by the underlying SQL database. The consequences can range from unauthorized viewing of data to database modification, deletion, or even issuing commands to the operating system in severe cases.

5.1.1 Consequences of a SQL Injection Attack

The outcomes of a successful SQL injection attack can be severe and varied, including:

- **Compromised Credentials**: attackers may acquire user credentials through SQL injection, allowing them to masquerade as legitimate users and exploit their access rights.
- **Database Breach**: attackers can gain unauthorized entry to database servers, exposing sensitive data stored within.
- **Data Tampering**: attackers have the capability to modify or insert new data into the compromised database, potentially undermining data integrity.
- **Data Erasure**: attackers can remove records from the database or even eliminate entire tables, leading to significant data loss.
- **Expanding Access**: by obtaining operating system privileges on database servers, attackers can leverage these permissions to infiltrate other critical systems within the network.

5.1.2 Types of Attack

SQL injection can manifest in various forms, each exploiting different vulnerabilities in a database's SQL implementation [5]:

- **Union-Based SQL Injection**: this is the most common form of SQL injection, utilizing the UNION SQL operator to combine the results of two or more SELECT statements into a single result set. This technique allows attackers to retrieve data from other tables within the database.
- **Error-Based SQL Injection**: this technique specifically targets MS-SQL Servers. It involves inducing the database to generate an error message. Attackers formulate queries that, when improperly handled by the database, result in error messages that inadvertently disclose sensitive information.

- **Blind SQL Injection**: unlike the methods mentioned above, Blind SQL Injection does not rely on the database returning error messages. Instead, attackers make logical guesses about the database information and observe the application's response to determine if their guess was correct. This category further splits into:
  - **Boolean-Based SQL Injection**: where the attacker sends a SQL query to the database, which forces the application to return a different result depending on whether the query returns true or false.
  - **Time-Based SQL Injection**: this method relies on sending SQL queries to the database which cause the database to wait for a specified amount of time before responding. The delay in response time indicates to the attacker whether the query condition was true or false.

5.1.3 Typical attack

Consider the scenario where the code below is used to fetch items based on the current user's username and a specified item name:

```
...
string userName = ctx.getAuthenticatedUserName();
string query = "SELECT * FROM items WHERE owner = '"
                + userName + "' AND itemname = '"
                + ItemName.Text + "'";
...
```

This code segment is vulnerable due to its reliance on concatenation to construct the SQL query. When the inputs for the username and item name are concatenated, the resulting SQL query is:

```
SELECT * FROM items
WHERE owner =
AND itemname = ;
```

If an attacker inputs the string `Widget' OR 'x'='x` for `itemname`, the constructed SQL command becomes:

```
SELECT * FROM items
WHERE owner = 'John'
AND itemname = 'Widget' OR 'x'='x';
```

This is effectively equivalent to executing:

```
SELECT * FROM items;
```

This query causes the database to return all rows in the `items` table, allowing the attacker to bypass the intended data access restrictions and retrieve sensitive information.

5.1.4 Mitigating SQL Injection Vulnerabilities

Mitigating the risk of SQL injection requires a multi-faceted approach:

- **Input Validation and Sanitization**: ensure all user inputs are validated against a strict specification of what is considered secure and sanitize inputs to remove or neutralize character sequences that could be interpreted as SQL commands.
- **Use of Prepared Statements and Parameterized Queries**: these SQL statement types separate SQL logic from the data, making it virtually impossible for an attacker to inject malicious SQL through user inputs.
- **Least Privilege Access**: limit the database access rights for applications to only those that are absolutely necessary for their function. This reduces the potential impact of a successful injection attack.
- **Regular Security Audits and Vulnerability Scanning**: regularly test web applications for vulnerabilities, particularly SQL injection, and promptly address identified issues.
- **Educating Developers**: ensure developers are aware of the risks associated with SQL injection and are knowledgeable in secure coding practices.

5.2 Denial of Service (DoS) Attacks

Denial of Service (DoS) [50] attacks constitute a significant threat to database security, aiming to disrupt the normal operations of database systems by overwhelming them with a flood of requests. Unlike other cyber attacks that seek unauthorized access or data theft, DoS attacks focus on rendering database services unavailable to legitimate users, thereby causing operational disruptions and potential financial losses.

DoS attacks exploit the limited capacity of databases to handle concurrent connections and requests. By inundating a database with more traffic than it can process, attackers can effectively halt its operations. These attacks can be direct, targeting the database itself, or indirect, affecting the infrastructure and network components that support database connectivity.

5.2.1 Types of DoS Attacks

- **Volume-based Attacks**: aim to consume the bandwidth of the target network or server, including the database server, with a sheer volume of traffic.
- **Protocol Attacks**: exploit weaknesses in the transport and application layer protocols to consume server resources, leading to service degradation or shutdown.
- **Application Layer Attacks**: directly target the application logic of database-driven applications, exhausting resources by initiating complex database queries that consume excessive processing power.

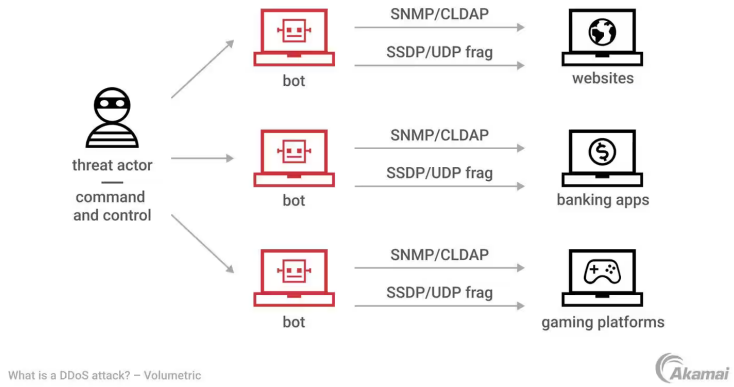


Figure 7: DOS Attack Schema

Source: `https://www.akamai.com/glossary/what-is-ddos`

5.2.2 Impact of DoS Attacks

The consequences of a successful DoS attack on a database can range from temporary service unavailability to prolonged downtime. These disruptions can impact business operations, damage customer trust, and incur significant recovery costs. For databases that support critical infrastructure or services, the impact can extend beyond financial loss, potentially affecting public safety or security.

Beyond operational disruptions and financial losses, Denial of Service attacks on databases can have a wider array of impacts that extend to various facets of an organization and its stakeholders. These additional consequences underscore the multifaceted nature of the threat posed by DoS attacks:

- **Degradation of Customer Experience and Trust**: A direct consequence of database unavailability is the degradation of customer experience. Customers expecting reliable access to online services, such as e-commerce platforms, banking applications, or informational websites, are met with delays or inability to access these services. Repeated or prolonged incidents can erode trust, leading to a loss of customer loyalty and potentially driving users to competitor services.
- **Legal and Regulatory Implications**: For organizations subject to data protection regulations and service-level agreements (SLAs), DoS attacks can lead to non-compliance issues. Regulations such as GDPR in Europe mandate certain levels of service availability and data accessibility. Failure to meet these obligations due to a DoS attack could result in legal actions, regulatory fines, and mandated reporting of the incident to authorities and affected parties.
- **Impact on Brand Reputation**: The public perception of a company can be significantly damaged by high-profile DoS attacks. The visibility of such attacks, especially when covered by media, can tarnish a brand's image, portraying it as unreliable or insecure. This reputational damage can be long-lasting and more costly to repair than the immediate financial or operational impacts of the attack.

- **Resource Diversion**: Responding to and mitigating a DoS attack requires significant organizational resources. IT and security teams must divert their focus from other critical tasks to address the immediate threat, potentially delaying projects and strategic initiatives. The cost of mitigation, including employing additional protective measures and technology to thwart future attacks, can also be substantial.

- **Competitive Disadvantage**: In highly competitive markets, any downtime can provide an edge to competitors. DoS attacks can create openings for competitors to capture market share by offering uninterrupted services. This is particularly impactful when attacks coincide with critical business periods, such as sales events, product launches, or financial reporting periods.

5.2.3 Mitigation Strategies

To safeguard databases against DoS attacks, organizations must adopt a multi-layered defense strategy:

- **Traffic Filtering**: implementing advanced traffic filtering solutions to detect and block malicious traffic before it reaches the database server.

- **Rate Limiting**: applying rate limiting on database requests to prevent excessive load from a single source or multiple coordinated sources.

- **Scalability and Redundancy**: designing database architecture for scalability and redundancy can help absorb unexpected surges in traffic, ensuring availability even under attack.

- **Monitoring and Alerting**: continuously monitoring database performance and network traffic for anomalies indicative of a DoS attack, coupled with an effective alerting system, can facilitate rapid response and mitigation.

- **Incident Response Plan**: developing and regularly updating an incident response plan specific to DoS attacks ensures preparedness and swift recovery from such incidents.

5.3 Insider Threats

Insider threats represent a critical security risk, stemming from individuals within an organization who have legitimate access to databases and sensitive information. Unlike external attacks, insider threats are challenging to detect and mitigate due to the attackers' legitimate access rights and knowledge of the organization's systems and processes. These threats can originate from employees, contractors, or anyone else who has been entrusted with access to the organization's data.
Insider threats can manifest in various forms, including intentional breaches where an insider deliberately steals, leaks, or sabotages data. However, not all insider threats are malicious; many result from negligence or ignorance, such as an employee accidentally exposing data through carelessness or a lack of awareness of security protocols.
As mitigation strategies, we can adopt the following strategies and policies:

- **Least Privilege Access**: implement and enforce the principle of least privilege, ensuring that individuals have only the minimum level of access necessary for their roles.
- **Continuous Monitoring and Behavioral Analysis**: use advanced monitoring tools to detect unusual patterns of behavior that may indicate malicious activity or misuse of access.
- **Training and Awareness**: regularly educate employees about data security practices, the importance of safeguarding data, and the potential consequences of data breaches.
- **Incident Response and Insider Threat Programs**: develop a comprehensive insider threat program that includes clear policies, procedures for reporting suspicious activity, and an effective incident response plan.

5.3.1 Malware and Ransomware

Malware and ransomware pose significant threats to database security, capable of corrupting, stealing, or encrypting data. Malware encompasses various forms of malicious software designed to infiltrate, damage, or take control of systems, while ransomware is a specific type of malware that encrypts the victim's data, demanding payment for its release. They operate:

- **Infiltration**: malware can enter an organization's network through phishing emails, compromised websites, or unpatched vulnerabilities in software. Ransomware often follows a successful malware infection, leveraging the same vulnerabilities or using malware to distribute the ransomware.
- **Data Corruption and Theft**: once inside the network, malware can be used to corrupt databases, exfiltrate sensitive information, or facilitate further attacks.
- **Encryption and Ransom Demand**: ransomware encrypts data, rendering it inaccessible to users. Victims are then presented with a ransom demand, typically requiring payment in cryptocurrency, for the decryption key.

5.3.2 Defense Mechanisms

Real defense mechanisms don't exist. For fighting this attack, prevention is the key word:

- **Up-to-Date Security Software**: ensure that all systems are protected with up-to-date antivirus and anti-malware solutions capable of detecting and neutralizing threats.
- **Regular Backups**: maintain regular, encrypted backups of critical data, stored separately from the main network, to ensure data recovery in the event of a ransomware attack.
- **Vulnerability Management**: regularly patch and update systems and software to close off vulnerabilities that could be exploited by malware and ransomware.

- **Security Awareness Training**: educate employees on the risks of phishing and social engineering attacks, which are common vectors for malware distribution.

5.4 Data Leakage and Unauthorized Access

Data leakage and unauthorized access are some of the main concerns in database security, often leading to the exposure of **sensitive information**. These vulnerabilities can arise from various sources, including poorly secured APIs, inadequate authentication mechanisms, and the absence of data encryption. Understanding and mitigating these risks are essential for safeguarding data integrity and privacy.

5.4.1 Exposure through APIs

Application Programming Interfaces (APIs) serve as gateways for applications to interact with databases, fetching or modifying data based on requests from users or other applications. However, vulnerabilities in API implementations can become conduits for data exposure. Insufficient input validation, lack of rate limiting, and insecure endpoints can allow attackers to exploit APIs, leading to unauthorized data access or leakage. As mitigation strategies:

- Implement rigorous input validation and sanitization to prevent injection attacks through API requests.
- Employ rate limiting to thwart brute-force attacks and reduce the efficacy of automated data extraction attempts.
- Secure API endpoints with authentication and employ HTTPS to encrypt data in transit, ensuring that only authorized requests are processed.

5.4.2 Poor Authentication Mechanisms

Authentication mechanisms are the first line of defense against unauthorized access. Weak or improperly implemented authentication processes can easily be bypassed by attackers, granting them access to sensitive data or administrative functions. Common issues include reliance on simple passwords, lack of multi-factor authentication (MFA), and vulnerabilities in authentication protocols. As mitigation strategies:

- Enforce strong password policies and consider implementing passwordless authentication methods that rely on more secure factors.
- Utilize multi-factor authentication (MFA) to add an additional layer of security, significantly reducing the risk of unauthorized access.
- Regularly audit and update authentication protocols to address any known vulnerabilities and ensure they align with current security standards.

5.4.3 Lack of Data Encryption

Data encryption is crucial for protecting the confidentiality of data both at rest and in transit. The absence of encryption, or the use of outdated encryption methods, can leave data vulnerable to interception and unauthorized access. Unencrypted

data can be easily read by anyone who gains access, whether through a network interception, physical access to storage media, or bypassing application security controls. As mitigation strategies:

- Implement encryption for data at rest using strong encryption standards to ensure that stored data is unintelligible without the proper decryption keys.
- Ensure that data in transit is protected using protocols such as HTTPS, TLS, or VPNs, which encrypt data as it travels across networks, preventing eavesdropping or interception.
- Manage encryption keys securely, using services like Azure Key Vault, to prevent unauthorized access to the keys themselves.

5.5 Configuration and Operational Flaws

Operational integrity and the security posture of databases are often compromised not just by external threats but also by internal configuration and operational flaws. These vulnerabilities stem from oversight in the initial setup, maintenance, and policy enforcement stages of database management. Addressing the risks associated with default settings, inadequate patch management, and improper access controls is crucial for safeguarding databases against potential breaches.

5.5.1 Default Settings

Many databases come with default configurations and credentials intended to simplify initial setup processes. However, these defaults are widely known and can serve as easy entry points for attackers. Risks associated with using default database configurations and credentials include unauthorized access and potential data breaches, as attackers exploit well-known defaults to gain entry into the system. As mitigation strategies:

- Change default usernames and passwords before deploying any database system into production.
- Review and customize default configuration settings to ensure they align with the security needs of the organization, disabling unnecessary services or features that could introduce vulnerabilities.

5.5.2 Inadequate Patch Management

Software vulnerabilities are a fact of life, and vendors regularly release patches and updates to address newly discovered security flaws. The delay or failure in applying these security patches and updates can leave databases exposed to exploits targeting known vulnerabilities, potentially allowing attackers to compromise the system. As mitigation strategies:

- Establish a regular schedule for checking and applying updates and patches to database software and associated applications.
- Automate the patch management process where possible to ensure timely application of critical security updates.

- Subscribe to security bulletins or services that provide alerts on new vulnerabilities and patches for database systems.

5.5.3 Improper Access Controls

Access controls are fundamental to database security, ensuring that only authorized users can access or modify data. Improper or poorly implemented access control policies can lead to unauthorized data access, data manipulation, or even system takeover. Common issues include overly permissive access rights, lack of role-based access controls, and failure to regularly review access permissions. As mitigation strategies:

- Implement principle of least privilege, ensuring users are granted only the access rights necessary for their roles.
- Utilize role-based access control (RBAC) to manage and enforce access policies efficiently.
- Conduct periodic audits of access controls and user privileges to identify and rectify overly permissive access or other anomalies.

Chapter 6

Microsoft Azure's Defensive Strategies for Database Security

Microsoft Azure offers a comprehensive suite of defensive strategies designed to protect databases from a wide array of threats: security solutions, focusing on firewalls, intrusion detection and prevention systems (IDPS), and identity management are nowadays a must-have in any organization's approach to data security.

6.1 Azure Firewall

Securing network boundaries is essential for safeguarding databases against unauthorized access and cyber threats. Microsoft Azure offers a robust solution in the form of Azure Firewall, a **stateful firewall service** with built-in high availability and unrestricted cloud scalability [41]. It provides network traffic filtering across **Azure Virtual Networks**, enabling users to prevent unauthorized access while allowing legitimate traffic to pass. Designed to protect Azure resources, Azure Firewall is a critical component in a comprehensive cloud security strategy, offering centralized network traffic control, threat intelligence, and application-level protection. By leveraging Azure Firewall alongside other Azure security features, organizations can create a robust defense mechanism against various cyber threats. Its key features comprehend:

- **Stateful Inspection**: Azure Firewall performs stateful packet inspection, tracking the state of active connections and making decisions based on the context of the traffic, which enhances its ability to block or allow traffic accurately.
- **Application Rule Collections**: it allows administrators to create rules that govern access to applications by filtering outbound HTTP/S traffic to specified fully qualified domain names (FQDNs), ensuring that only approved applications can be accessed.
- **Network Rule Collections**: these rules control traffic flow between subnets, enforcing policies at the network layer. They specify source and destination addresses, protocols, port numbers, and actions (allow or deny), offering granular control over network traffic.
- **Threat Intelligence**: integrated with Microsoft Threat Intelligence, Azure

Firewall automatically identifies and blocks known malicious IP addresses and domains, providing an additional layer of security against cyber threats.

- **FQDN Tagging**: this feature simplifies the creation of network rules for Azure services by allowing the use of service tags instead of specifying individual IP addresses, making rule management more efficient.

6.1.1 Implementing Azure Firewall for Database Protection

When deploying Azure Firewall to protect Azure databases, administrators should consider the following practices:

- **Segmentation and Micro-Segmentation**: utilize Azure Firewall to create network segments and micro-segments that isolate database resources from other network resources, minimizing the potential impact of breaches.
- **Custom Application Rules**: define custom application rules that specifically cater to the database environment, ensuring that only traffic necessary for database operations is allowed.
- **Continuous Monitoring and Updating**: regularly review and update firewall rules in response to changes in the network environment or emerging threats. Leverage Azure Firewall's threat intelligence and logging capabilities to identify and respond to potential security incidents.

One of the most straightforward and effective measures to enhance database security is to restrict access to the database server based on IP addresses [42]. This can be achieved through server-level IP firewall rules that define which IP addresses or ranges are allowed to communicate with the server. For instance, it is possible to create a rule that only allows traffic from the IP addresses of your corporate office and VPN, effectively blocking all other unauthorized attempts to access the database. This can be achieved through Azure Portal [43], PowerShell. Same goal can be achieved using Transact-SQL:

```
EXECUTE sp_set_firewall_rule @name = N'CorporateOfficeAccess',
   @start_ip_address = '203.0.113.0',
   @end_ip_address = '203.0.113.255';
```

Another Firewall feature to use to enhance security is to leverage Azure Firewall's application rule collections. These enable administrators to define access rules based on Fully Qualified Domain Names (FQDNs). This is particularly useful for controlling outbound traffic from the database server to specific domains, ensuring that the server can only communicate with authorized external services. For example, could be useful and a good security practice to create separate network segments for development, testing and production databases to prevent unauthorized access or data leakage. Firewall rules can be tailored for each segment based on specific access needs and risk profile.

6.2 Identity Management: Fortifying Access Control

Central mechanism of Database protection is Microsoft EntraID, a comprehensive identity and access management service. EntraID's suite of features empowers organizations to implement stringent authentication and authorization protocols [23],

effectively fortifying access control to Azure databases.

6.2.1 Multi-Factor Authentication (MFA)

Multi-Factor Authentication (MFA) introduces an indispensable layer of security. EntraID's MFA requires users to verify their identity through **multiple pieces of evidence** (or factors) before granting access to database resources. This could include **something they know** (a password), **something they have** (a smartphone app or token), or **something they are** (biometrics).

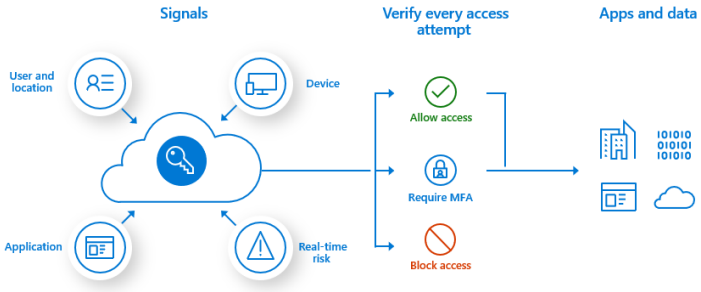


Figure 8: Multi Factor Authentication on Microsoft EntraID
Source: `https://learn.microsoft.com/en-us/entra/identity/authentication/concept-mfa-howitworks`

Enabling MFA for Azure databases involves configuring EntraID to prompt for additional authentication methods beyond just the user's password [24]. This approach significantly reduces the likelihood of unauthorized access, even in the event of credential theft.

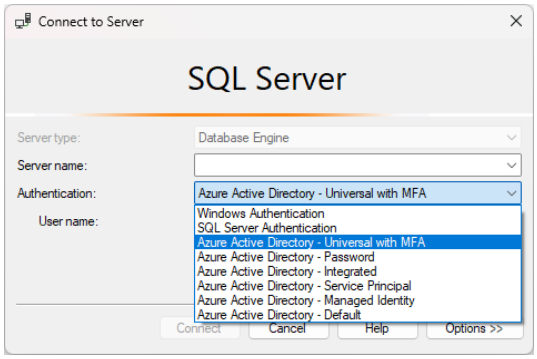


Figure 9: Authentication on SQL Server Management Studio (SSMS)

6.2.2 Conditional Access Policies

Conditional Access Policies in EntraID allow for the specification of conditions under which access to Azure databases is permitted [25]. These policies **evaluate several signals**, including user **location**, **device compliance**, and **risk levels**, to dynamically grant or block access attempts. Specifically for Azure SQL Database,

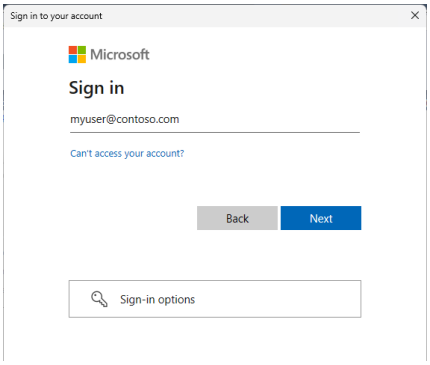


Figure 10: MFA on Azure SQL with EntraID MFA Enabled

Conditional Access policies can help ensure that only authenticated and authorized users can access the databases under specific conditions, enhancing security.

Administrators can configure policies to restrict database access to users connecting using compliant, managed devices. This granularity ensures that access is securely controlled, adapting to the context of each access attempt.

6.2.3 Role-Based Access Control (RBAC)

Azure's implementation of Role-Based Access Control (RBAC) is integral to its identity management strategy. RBAC ensures that access rights are aligned with the user's role within the organization, granting them only the permissions necessary to perform their duties. This minimizes the risk of unauthorized access or data exposure by adhering to the principle of least privilege.

Assigning roles in EntraID, such as **database reader**, **writer**, or **administrator**, allows for precise control over who can access and perform operations on database resources. This segmentation of roles and permissions is pivotal in maintaining a secure and organized access control system. Some of them are pre-defined and available for administrators [44]:

- **Contributor**: provides comprehensive management capabilities for all resources, excluding role assignments in Azure RBAC, Azure Blueprints assignments management, and sharing of image galleries.
- **Owner**: offers complete control over all resources, inclusive of the capability to assign roles within Azure RBAC.
- **Reader**: allows viewing of all resources without the ability to perform modifications.
- **RBAC Administrator**: empowers the assignment of roles to manage Azure resource access strictly through Azure RBAC, without managing access via alternative methods like Azure Policy.
- **User Access Administrator**: enables management of user access permissions for Azure resources.

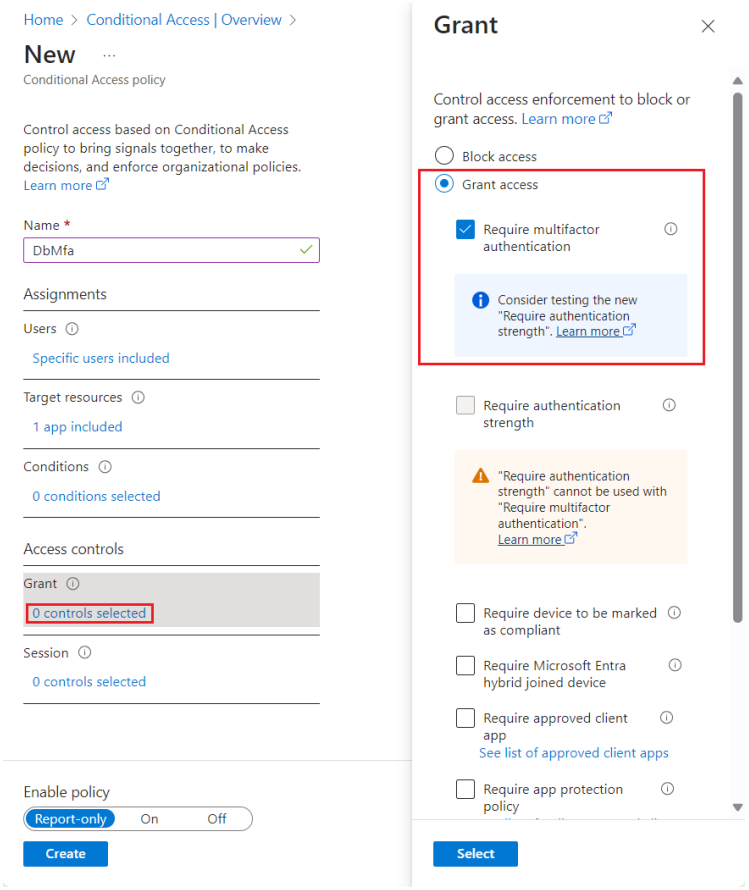


Figure 11:

Figure 12: Conditional Access configuration on Azure SQL example
Source: `https://learn.microsoft.com/en-us/azure/azure-sql/database/conditional-access-configure?view=azuresql`

6.3 Intrusion Detection and Prevention Systems (IDPS)

Azure's IDPS solutions play a crucial role in **identifying**, **blocking**, and **mitigating threats** in real-time, ensuring the integrity and availability of database resources. Azure's IDPS capabilities are integral components of its security framework, designed to monitor network and system activities for malicious actions or policy violations. These systems not only detect a wide array of threats, including unauthorized access attempts and exploitation of vulnerabilities but also take predefined actions to prevent or mitigate potential damage.

- **Detection**: leveraging a comprehensive set of signatures and anomaly detection algorithms, Azure IDPS continuously monitors traffic and activities, identifying patterns indicative of cyber threats.
- **Prevention**: upon detection of a potential threat, the system can automatically take action to block the threat from causing harm, such as terminating malicious connections or quarantining affected resources.

- **Alerting**: administrators are immediately alerted to potential security incidents, allowing for rapid response and remediation efforts.

6.3.1 Key Components of Azure's IDPS

Azure's IDPS [9] ecosystem comprises several key components, each contributing to the overall security posture of Azure-hosted databases:

- **Microsoft Defender For Cloud**: at the forefront of Azure's IDPS capabilities, Microsoft Defender For Cloud offers advanced threat protection, including continuous monitoring and threat intelligence-based detection.
- **Azure Sentinel**: as a cloud-native SIEM service, Azure Sentinel provides intelligent security analytics across the enterprise, enhancing the ability to detect, investigate, and respond to threats.
- **Azure Advanced Threat Protection (ATP)**: specifically designed to identify, detect, and help investigate advanced threats, compromised identities, and malicious insider actions directed at your organization.

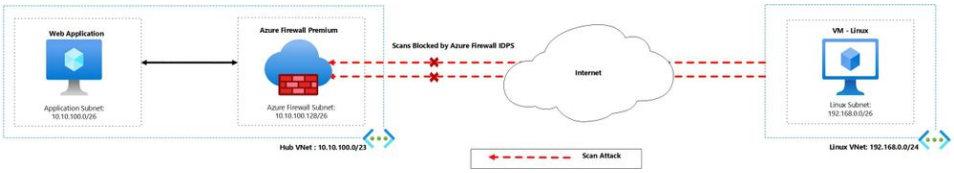


Figure 13: Example of Azure IDPS mitigating a Scanning Attack
Source: `https://techcommunity.microsoft.com/t5/azure-network-security-blog/intrusion-detection-and-prevention-system-idps-based-on/ba-p/3921330`

6.3.2 Implementing Azure IDPS for Database Security

To effectively leverage Azure's IDPS for database security, organizations should:

- **Enable Advanced Threat Protection**: activate ATP in **Microsoft Defender For Cloud** to automatically monitor SQL databases for unusual and potentially harmful attempts to access or exploit databases [26].
- **Configure Security Policies**: define and apply security policies that reflect the organization's risk tolerance and compliance requirements, ensuring that IDPS mechanisms are tuned to detect and respond to relevant threats.
- **Review and Act on Alerts**: regularly review alerts generated by IDPS tools and take prompt action to investigate and mitigate identified threats, minimizing potential impacts on database integrity and availability.

Following, the best practices for ensuring that Azure IDPS are working as best as possible:

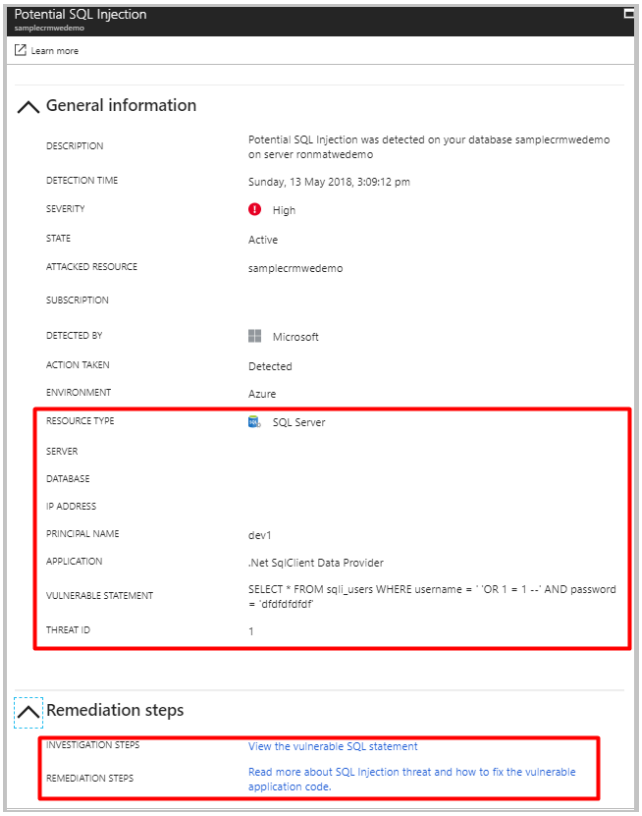


Figure 14: Example of Azure Advanced Threat Protection intercepting a potential SQL Injection Attack
Source: `https://learn.microsoft.com/en-us/azure/azure-sql/database/threat-detection-overview?view=azuresql`

- **Regularly Update Signatures and Definitions**: ensure that IDPS components are up-to-date with the latest threat intelligence for effective detection of new and evolving threats.
- **Integrate IDPS with Incident Response Plans**: incorporate IDPS alerts and data into your broader incident response plan, ensuring a coordinated and effective response to detected threats.
- **Continuous Monitoring and Assessment**: continually assess the effectiveness of your IDPS setup, adjusting configurations as needed to adapt to changing threat landscapes and organizational needs.

Intrusion Detection and Prevention Systems are critical elements of Azure's comprehensive approach to database security, providing the means to detect, prevent, and respond to cyber threats effectively. By integrating Azure's IDPS capabilities into their security strategy, organizations can achieve a higher level of protection for their cloud-based databases, safeguarding valuable data assets against a wide spectrum of cyber threats.

6.3.3 Machine Learning in Azure Threat Detection

The integration of **machine learning** (ML) into threat detection represents a significant step forward in the cybersecurity domain, especially in the context of database

security. By harnessing the power of ML, organizations can enhance their capabilities to identify, mitigate, and rectify database threats with unprecedented precision and efficiency. Traditionally, threat detection systems relied heavily on signature-based methods that required prior knowledge of attack patterns. While effective against known threats, these methods struggled to keep pace with the rapidly evolving landscape of cyberattacks, particularly those not yet documented or entirely new (**zero-day attacks**). Machine learning introduces a paradigm shift [54], enabling proactive and dynamic detection of both known and unknown threats by learning from data rather than relying solely on predefined signatures.

Azure leverages machine learning across its security services to provide advanced threat detection capabilities that continually evolve and adapt. Microsoft Defender For Cloud, Azure Sentinel, and Azure Advanced Threat Protection (ATP) all utilize ML algorithms to analyze vast quantities of data, identifying anomalies that could indicate a threat to databases.

- **Anomaly Detection**: ML models are trained to understand normal database activities and configurations. Deviations from these baselines are flagged for further investigation, enabling early detection of potential threats before they manifest into breaches.
- **Predictive Analytics**: by analyzing historical data, ML models can predict potential future attacks, allowing organizations to preemptively strengthen their defenses in vulnerable areas.
- **Automated Threat Intelligence**: ML enhances threat intelligence by automatically correlating threat data from diverse sources, providing a more comprehensive understanding of potential attack vectors and vulnerabilities.

The application of machine learning in database security within Azure offers several key advantages:

- **Real-time Detection**: ML algorithms continuously monitor database activities, enabling real-time detection of suspicious behavior, unauthorized access attempts, or abnormal data transactions.
- **Mitigation Strategies**: upon detection of a potential threat, ML-driven systems can automatically initiate predefined mitigation strategies, such as temporarily restricting access, alerting administrators, or isolating affected database segments.
- **Incident Analysis and Fix**: post-detection, ML algorithms assist in analyzing the incident to understand the attack's nature, scope, and impact. This analysis supports the development of fixes or patches to prevent recurrence and strengthen database security postures.

Chapter 7

Protection in practice: the development of a secure .NET Library

The initial segment of this thesis is designed as an introductory guide for those venturing into the vast and complex domain of data and its security within the Azure platform. The Azure ecosystem, with its plethora of services and tools dedicated to data management and protection, can initially appear daunting due to the sheer volume of data and information to be absorbed. This situation poses a particularly challenging environment for individuals and organizations aiming to explore and leverage the capabilities of .NET in conjunction with Azure, encompassing aspects related to databases and data security.

Transitioning to the use of cloud platforms like Azure for application development, especially when integrating .NET, databases, and data protection measures, introduces a steep learning curve. Newcomers might find themselves initially disoriented, grappling with the intricacies of cloud-based development and the myriad of services Azure offers. This phase can be long, arduous, and costly, requiring a significant investment of time and resources to achieve proficiency.

This part of the thesis aims to serve as a bridge, guiding readers through the initial complexities of Azure and .NET, smoothing the learning curve, and shedding light on cost-effective and efficient strategies for data security and management in the cloud. It is crafted with the hope of demystifying the initial stages of Azure-based development for those new to the platform, providing a solid foundation upon which they can build their skills and knowledge in data security and .NET development within Azure.

7.1 The project: DotNetArmor

The purpose of the project is to offer **small and medium-sized enterprises** (SMEs) **and developers**, especially those new to .NET and database management, essential tools for immediately enhancing productivity with .NET and databases in a secure manner. This focus allows users to concentrate more on the development of their application rather than on the intricacies of security. This is particularly important since the array of Azure services dedicated to security is vast and can be quite expensive, which may be daunting, especially at the initial stages where the

volume of data doesn't justify significant expenditure. In an environment filled with frequently changing and numerous services, it's easy to lose one's way.

The project consists of a .NET library called **DotNetArmor** that provides a secure and comprehensive data access layer. Through the integration of security libraries and data management techniques, it ensures the sanitization of inputs and outputs, addresses data access, and crucially, includes an endpoint to be used before queries. This endpoint checks whether user-supplied input contains malicious strings, offering a first line of defense against SQL injection and other database-related security threats.

Furthermore, the library is enriched with additional security tools, not all of which are solely focused on database security but on general application security. These enhancements are designed to mitigate further threats such as anti-forgery, cross-site scripting (XSS), and more. This holistic approach to security, while maintaining a user-friendly interface for developers, ensures that SMEs can deploy .NET applications that are not only efficient and productive but also secure from a broad spectrum of potential vulnerabilities.

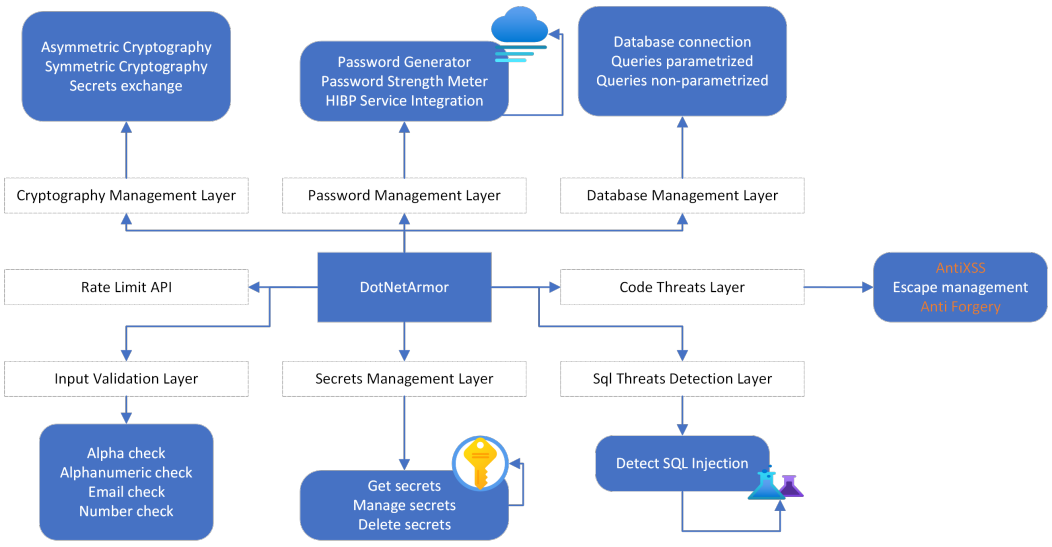


Figure 15: DotNetArmor library architecture: tools and services.

7.2 Library Specifics

- **Secure Password Generation**: the library introduces a robust system for generating strong, secure passwords. Utilizing advanced algorithms, it ensures the creation of passwords that resist brute-force attacks and meet security standards for complexity.
- **Input Sanitization and Anti-XSS**: input sanitization is critical in safeguarding applications from malicious input. The library offers comprehensive sanitization features, effectively neutralizing potential **cross-site scripting** (XSS) threats. It ensures that incoming data is properly cleaned, allowing only safe input to pass through.

- **Cryptography Services**: the library boasts an extensive range of cryptographic services:
  - **3DES and AES**: for symmetric encryption, it supports Triple Data Encryption Standard (3DES) and Advanced Encryption Standard (AES), catering to varying levels of security and performance needs.
  - **RSA**: asymmetric encryption is facilitated through RSA, providing a secure means of data exchange over public networks.
- **Hashing (SHA)**: it includes hashing functions such as **SHA-256** and **SHA-512**, essential for data integrity verification and secure storage of sensitive information.
- **Database Management**: Database interactions are streamlined through the library's ability to perform SQL queries securely. It supports the execution of queries, updates, and transactions, with a strong focus on **preventing SQL injection attacks**.
- **Anti-DOS and API Rate Limitation**: to fight Denial-of-Service (DOS) attacks, the library incorporates API rate limiting functionalities. This allows applications to define and enforce limits on the number of requests a user can make, preserving resources and ensuring availability.
- **Secret Sharing (Diffie-Hellman)**: the library implements the Diffie-Hellman algorithm for secure secret sharing. This feature enables two parties to establish a shared secret over an insecure channel, a cornerstone for encrypted communication.
- **Key Vault Management**: secure management of secrets and keys is critical in maintaining the confidentiality and integrity of data. The library provides mechanisms for interacting with services like Azure Key Vault, allowing for the secure storage, retrieval, and rotation of cryptographic keys and secrets.

7.3 Secret Management

Azure Key Vault [62] is a cloud service provided by Microsoft that offers secure storage of cryptographic keys, secrets, tokens, and certificates. It's designed to safeguard cryptographic keys and other secrets used by cloud applications and services. By centralizing the storage of these sensitive items, Azure Key Vault simplifies the task of key management, especially in complex environments. Incorporating Azure Key Vault into the `DotNetArmor.Security` namespace addresses a critical architectural need for secure and efficient key management, mitigating a previously identified weak point in managing cryptographic assets and credentials securely.

The `AzureKeyVaultService` class implements the `IAzureKeyVaultService` interface, providing a simplified and effective method for interacting with Azure Key Vault. This service enables applications to securely retrieve, store, and manage secrets, such as passwords, connection strings, or any piece of information that requires secure access.

This class implements the following methods:

- `SetKeyVaultConfiguration`: configures the service with the necessary Azure Key Vault parameters, including the vault URL and credentials for authentication.
- `GetSecretAsync`: asynchronously retrieves a secret from Azure Key Vault by its name.
- `SetSecretAsync`: asynchronously stores a new secret or updates an existing secret in Azure Key Vault.
- `DeleteSecretAsync`: asynchronously initiates the deletion of a secret from Azure Key Vault.

It brings some security advantages, including and not limited to:

- **Centralized Management**: facilitates centralized management of secrets, keys, and certificates, enhancing security and compliance.
- **Access Control**: provides granular access control policies, ensuring that only authorized applications and users can access stored secrets.
- **Audit Trails**: offers detailed audit trails, enabling monitoring and tracking of how and when secrets are accessed.
- **Automated Rotation**: supports the automated rotation of keys and secrets, helping to maintain a strong security posture over time.

Utilizing the Azure KeyVault service enhances security by eliminating the need to embed passwords, certificates, and secrets directly within the application's source code. Instead, when building an application or portal, we can integrate it seamlessly with Azure KeyVault. This integration allows us to insert merely a reference to our secrets in the code, rather than the sensitive information itself. For instance, should there be a requirement to reference a secret within the application, we can safely do so by storing only the reference to that secret.

```
9   "HibpApiKey": {
10    "Key": "@Microsoft.KeyVault(SecretUri=https://sqlthreat7456330756.vault.azure.net/secrets/hibp/6f75b38869d3457db5eb49412da37d49)"
11  }
12 },
```

Figure 16: A reference within the application to the KeyVault secret containing the API key for the third-party service *Have I Been Pwned?*.

This approach ensures that the actual secret remains concealed throughout the development process. Consequently, in the event that an attacker gains access to the source code, the secret remains secure and undisclosed.

During the development, these challenges and considerations have been found:

- **Proper Configuration**: ensuring the correct configuration of Azure Key Vault and its credentials is crucial for secure operations.
- **Network Reliability**: dependence on network connectivity to Azure services; ensuring high availability and reliability is essential.
- **Compliance and Policies**: adhering to compliance requirements and setting up appropriate access policies is vital for safeguarding secrets.

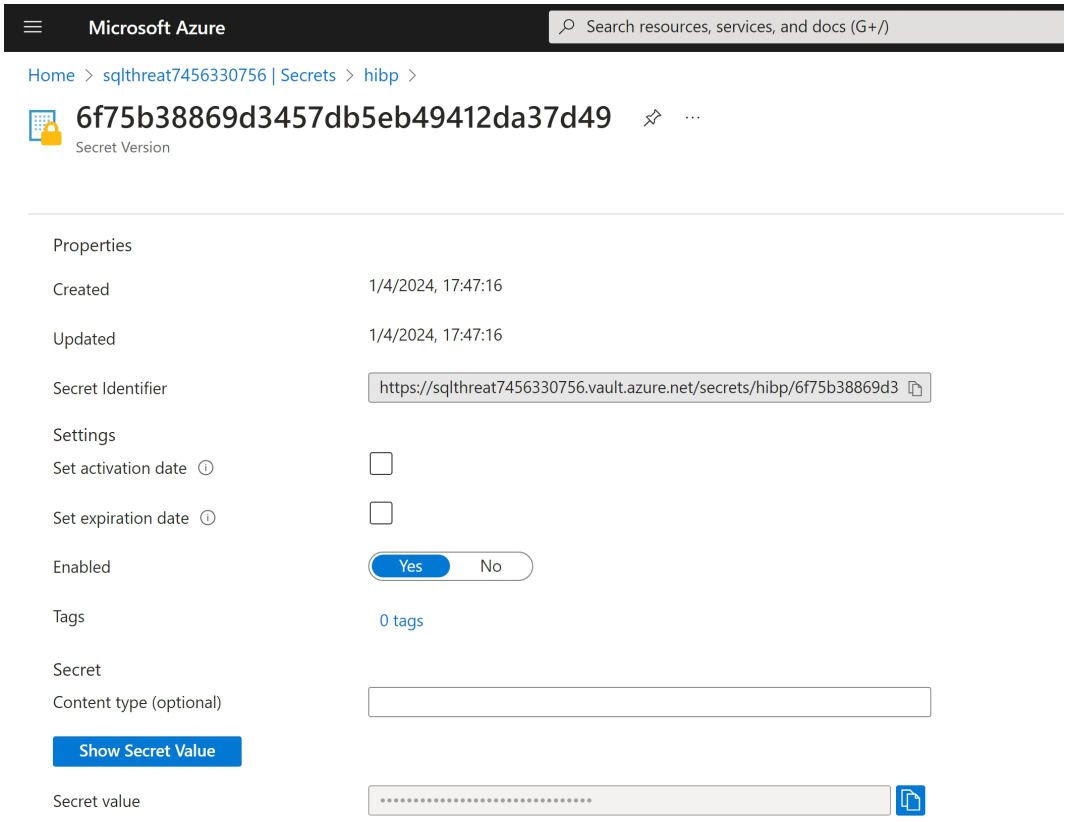


Figure 17: The real secret referenced in the application securely stored in Azure KeyVault. Note the same secret identifier (URI).

- **Cost Management**: monitoring and managing costs associated with Azure Key Vault usage, especially when handling a large number of secrets or high access rates.

7.4 Secure Password Management

The `DotNetArmor.Password` namespace provides a comprehensive suite of tools designed to enhance password security in .NET applications. The `PasswordService` class offers a range of functionalities essential for modern password management, including:

- **Secure Password Generation**: creates strong, random passwords based on specified criteria, enhancing security against brute-force attacks.
- **Password Strength Checking**: evaluates passwords against a set of criteria to determine their strength, encouraging the use of robust passwords.
- **Password and Email Breach Verification**: checks if a password or email has been exposed in known data breaches, leveraging the "Have I Been Pwned" (HIBP) API.
- **Salt Generation**: generates a cryptographic salt, improving the security of stored passwords by ensuring each password hash is unique.

This class implements the following methods:

- `GenerateSecurePassword(int length)`: generates a secure password of a specified length.
- `GenerateSalt(int size)`: creates a salt of a specified size.
- `GetHIPWApiKey(string apiKey)`: sets the API key required for HIBP API calls.
- `IsEmailPwnedAsync(string email)`: asynchronously checks if an email has been compromised in a data breach.
- `IsPasswordPwnedAsync(string password)`: asynchronously verifies if a password has appeared in a data breach.
- `CheckStrength(string password)`: assesses the strength of a password and categorizes it into levels such as "Very Weak", "Weak", "Medium", "Strong", or "Very Strong".

It brings some security advantages, including and not limited to:

- **Enhanced Security**: offers tools for generating strong passwords and salts, crucial for safeguarding user data.
- **Data Breach Awareness**: integrates with HIBP to provide real-time insights into whether user credentials have been compromised, enabling proactive security measures.
- **User-Friendly Security Practices**: the password strength checker encourages users to create stronger passwords, promoting better security habits.
- **Ease of Integration**: designed as a plug-and-play service within .NET applications, it simplifies the implementation of advanced password security measures.

During the development these challenges have been identified and addressed:

- **API Dependency**: relies on external services (HIBP) for some functionalities, which introduces a dependency on third-party API availability and its rate limits. This was addressed by using a third party library, **Polly** [3], which offers automatic incremental retries in case of API failure.
- **API Key Management**: requires management of API keys for HIBP, adding a layer of complexity and responsibility to ensure the security of these keys. This was addressed by relying on `Azure KeyVault`, a service that manages secrets, keys and secrets; it allows developers to reference to passwords instead of declaring them esplicitally in the code.
- **Performance Considerations**: real-time checks against HIBP for every login or password change can introduce latency, depending on network conditions and API response times.
- **Evolving Security Threats**: as security threats evolve, maintaining and updating the library to incorporate new security measures and best practices becomes necessary.

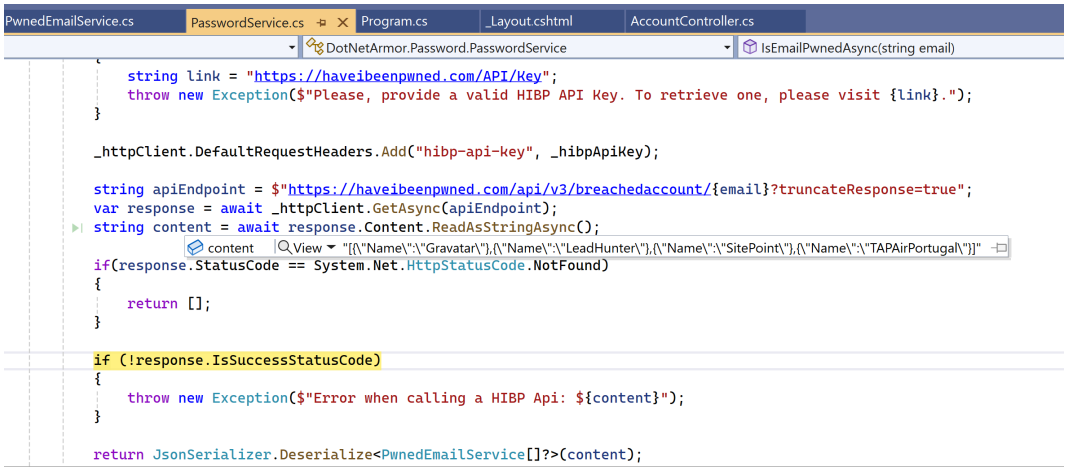


Figure 18: Third Party service (Have I Been Pwned?) that checks if the email has been found is some famous data breach (library's debug mode).

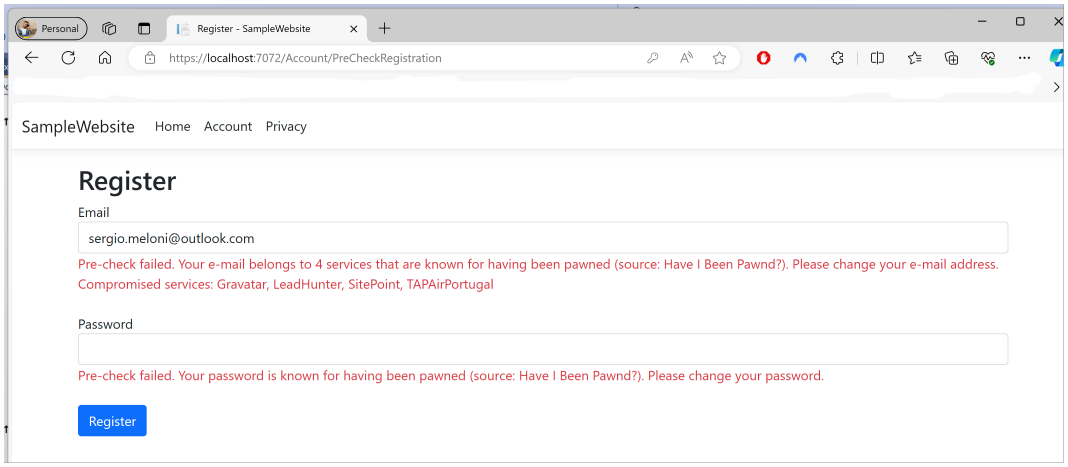


Figure 19: Third Party service (Have I Been Pwned?) that checks if the email has been found is some famous data breach (final result).

7.5 Hashing Manager

The `HashingService` class, part of the `DotNetArmor.Cryptography` namespace, is designed to provide various cryptographic hashing functionalities. It supports multiple hashing algorithms, including SHA-1, SHA-256, SHA-512, and Blake2, for both direct input data and file content.

This class implements the following methods:

- `ComputeSha1`: Computes the SHA-1 hash of the input data.
- `ComputeSha1256`: Computes the SHA-256 hash of the input data.
- `ComputeSha1256Checksum`: Computes the SHA-256 hash of a file's contents.
- `ComputeSha1512Checksum`: Computes the SHA-512 hash of a file's contents.

- `ComputeSha1512`: Computes the SHA-512 hash of the input data.
- `ComputeBlake2`: Applies the Blake2 hashing algorithm to the input data.

It brings some security advantages, including and not limited to:

- Ensuring data integrity by providing a unique hash value for given input, which can be used to verify the data's authenticity and unchanged state.
- Supporting a range of hashing algorithms allows for flexibility in security requirements and compatibility with various standards.
- Enhancing security for stored data, such as passwords, by utilizing advanced hashing functions like Blake2.

During the development, these challenges and considerations have been found:

- **Cryptographic Best Practices**: adhering to the latest security standards and best practices for cryptographic operations to ensure the highest level of security.
- **Performance vs. Security**: balancing the computational cost of hashing, especially for large files or data sets, with the need for robust security measures.
- **Hash Collision Resistance**: ensuring the chosen hashing algorithms remain resistant to collisions, where two different inputs produce the same output hash, which is critical for maintaining data integrity and security.
- **Deprecation of SHA-1**: considering the security vulnerabilities associated with SHA-1, there's a need to move towards more secure hashing algorithms like SHA-256 and SHA-512.

7.6 Symmetric Cryptography

The `DotNetArmor.Cryptography` namespace within the DotNetArmor library encompasses functionalities centered around cryptographic operations, with a focus on **AES** (Advanced Encryption Standard) encryption and decryption. The `AesEncryptionService` class offers an encapsulated approach to performing AES encryption and decryption, adhering to best practices in cryptography. It simplifies the process of encrypting and decrypting data, managing encryption keys, and initialization vectors (IVs) for consumers.

This class implements the following methods:

- `GenerateEncryptionKey()`: generates a new, random encryption key suitable for AES encryption. This key should be securely stored and managed.
- `GenerateInitializationVector()`: produces a new, random Initialization Vector (IV). Each encryption operation should use a unique IV to enhance security.
- `Encrypt(byte[] dataToEncrypt, byte[] key, byte[] iv)`: encrypts the provided data (dataToEncrypt) using the specified encryption key and iv, returning the encrypted data as a byte array. This method ensures data confidentiality.

- `Decrypt(byte[] dataToDecrypt, byte[] key, byte[] iv)`: decrypts the provided data (dataToDecrypt) using the specified decryption key and iv, returning the original data as a byte array. This method is the counterpart to Encrypt, allowing the recovery of the original data from its encrypted form.

It brings some security advantages, including and not limited to:

- **Security**: implements AES, a widely recognized encryption standard, ensuring a high level of security for data at rest or in transit.
- **Simplicity**: abstracts away the complexities involved in direct use of cryptographic libraries, providing a straightforward interface for encryption and decryption operations.
- **Flexibility**: by exposing both key and IV generation methods, it offers flexibility in managing cryptographic keys and IVs, which are critical for maintaining the security of encrypted data.

During the development these challenges and considerations have been found:

- **Key Management**: proper management of encryption keys is vital, like in the `PasswordService`. They should be stored securely in KeyVault and rotated periodically to maintain security.
- **IV Uniqueness**: each encryption operation should use a unique IV to prevent certain types of cryptographic attacks. The IV does not need to be kept secret but should be stored or transmitted alongside the encrypted data to allow for successful decryption.
- **Error Handling**: implementing appropriate error handling around cryptographic operations is crucial to manage exceptions and ensure the security and stability of applications utilizing this service. A complete logging system has been realized to address this issue.

The `AesEncryptionService` within the `DotNetArmor.Cryptography` namespace provides a robust and user-friendly mechanism for integrating AES encryption and decryption into .NET applications. By abstracting the intricacies of .NET's cryptographic model, it allows developers to more securely and easily manage sensitive data within their applications.

7.7 Asymmetric Cryptgraphy

The `RsaService` class, residing within the `DotNetArmor.Cryptography` namespace, encapsulates RSA encryption and decryption functionalities, as well as data signing and signature verification. This class implements the `IRsaService` interface, providing a robust set of methods for managing RSA keys, encrypting and decrypting data, and ensuring data integrity through digital signatures.

This class implements the following methods:

- `GenerateKeys`: generates a pair of RSA public and private keys with the specified key size.
- `Encrypt`: encrypts text using the provided RSA public key.

- `Decrypt`: decrypts text using the provided RSA private key.
- `SignData`: signs a message using the provided RSA private key to ensure its integrity and authenticity.
- `VerifySignature`: verifies the signature of a message using the corresponding public key, ensuring that the message has not been tampered with and was signed with the matching private key.

It brings some security advantages, including:

- Facilitating secure data transmission by providing strong encryption and decryption capabilities.
- Ensuring data integrity and non-repudiation through digital signatures and signature verification.
- Allowing for secure key generation and management practices.

During the development, these challenges and considerations have been found:

- **Key Management**: securely storing and managing RSA keys is crucial, as the security of RSA-based operations heavily relies on the confidentiality of the private key. For this, like for previous classes, Azure KeyVault is essential to protect secrets and passwords.
- **Performance Considerations**: RSA operations, especially with larger key sizes, can be computationally intensive, affecting the performance of applications.
- **Secure Key Generation**: ensuring the cryptographic strength of generated keys by using sufficient key sizes and secure random number generators.
- **Algorithm Flexibility**: while RSA is widely used, it's important to stay adaptable to newer algorithms and cryptographic practices as the landscape evolves.

7.8 Input Sanitization

The `InputSanitizeService` class, within the `DotNetArmor.Sanity` namespace, leverages the **AntiXSS** library [61] to provide a straightforward interface for sanitizing input across various contexts. By encapsulating the AntiXSS library's functionality, it offers a simplified approach to input encoding, enhancing security in .NET applications.

The AntiXSS library is a part of Microsoft's security application toolkit, designed to provide an array of methods for encoding data. This is crucial for protecting web applications from Cross-Site Scripting (XSS) attacks by sanitizing user input before it is rendered in web pages. AntiXSS replaces potentially dangerous characters with their HTML entity-encoded equivalents, thereby preventing malicious scripts from executing. The library supports encoding for HTML, JavaScript, URLs, CSS, XML, and more, making it a comprehensive solution for preventing XSS vulnerabilities in .NET applications.

This class implements the following methods:

- `HtmlEncode`: Encodes input strings for safe use in HTML content.
- `JavaScriptEncode`: Encodes input strings for safe use in JavaScript.
- `UrlEncode`: Encodes input strings for safe use in URLs.
- `HtmlAttributeEncode`: Encodes input strings for safe use in HTML attributes.
- `CssEncode`: Encodes input strings for safe use in CSS.
- `XmlEncode`: Encodes input strings for safe use in XML.
- `XmlAttributeEncode`: Encodes input strings for safe use in XML attributes.

It brings some security advantages, including and not limited to:

- **Comprehensive Protection**: offers a wide range of encoding options to safeguard against XSS in various contexts, including web pages, URLs, and CSS.
- **Ease of Use**: simplifies the process of encoding input, making it easier for developers to implement robust security measures.
- **Consistency**: provides a consistent approach to input sanitization across an application, reducing the risk of security oversights.

During the development, these challenges and considerations have been found:

- **Understanding Context**: it's crucial to understand the context in which data will be used to apply the appropriate encoding method. Incorrect encoding can either lead to security holes or unnecessarily mangled data.
- **Performance Impact**: while encoding operations are generally fast, extensive use on large volumes of data can impact application performance.
- **Keeping Up-to-Date**: as attack vectors evolve, staying updated with the latest security practices and library versions is essential.

7.9 Secret Sharing Management

Nowadays, establishing a shared secret over an insecure channel without exposing the secret itself is a cornerstone. The `DiffieHellmanService` class, part of the `DotNetArmor.SecretSharing` namespace, leverages the Diffie-Hellman key exchange algorithm to achieve this. The class facilitates the creation of a shared secret between parties, enabling encrypted communication that is resilient against eavesdropping.

The `DiffieHellmanService` implements the `IDiffieHellmanService` interface, providing methods to generate key pairs and compute shared secrets. It uses Elliptic Curve Diffie-Hellman (ECDH) for enhanced security and efficiency.

This class implements the following methods:

- `GenerateKeyPair`: generates an Elliptic Curve Diffie-Hellman (ECDH) key pair, returning both the public and private keys. The public key is shared with other parties, while the private key is kept secret.

- `GenerateSharedSecret`: computes a shared secret using the local private key and another party's public key. This shared secret can then be used to encrypt subsequent communications.

It brings some security advantages, including and not limited to:

- **Security**: the use of ECDH offers strong security even with shorter key lengths, making it computationally infeasible to derive the shared secret from public keys alone.
- **Efficiency**: Elliptic curve cryptography provides the same level of security as RSA with smaller keys, resulting in faster computations and less data to transmit during key exchange.
- **Flexibility**: this service can be integrated into various communication protocols to secure data exchange without requiring a secure channel for key distribution.

During the development, these challenges and considerations have been found:

- **Key Management**: proper handling and storage of private keys are crucial. Loss or exposure of a private key compromises the security of the shared secret.
- **Interoperability**: ensuring compatibility between different implementations of ECDH across various platforms and languages can be challenging.
- **Understanding Cryptography**: a solid understanding of cryptographic principles is necessary to appropriately utilize and configure the service for specific security requirements.
- **Algorithm Selection**: while ECDH is chosen for its efficiency and security, staying informed about advancements in cryptography is important to adapt to future security needs.

7.10 AntiDOS and Rate Limiting API

ASP.NET Core offers a robust solution for controlling the rate of requests to a web API or application. This capability is crucial for preventing excessive use or potential **Denial-of-Service** (DoS) attacks, thereby ensuring the application's availability and reliability. When integrated into DotNetArmor it enhances the application's defensive mechanisms by providing fine-grained control over incoming request rates.

Some of the advantages of ASP.NET Core Rate Limiting includes:

- **Anti-DoS Protection:** it plays a vital role in safeguarding applications against Denial-of-Service attacks by limiting the frequency of requests from a single source.
- **Resource Management:** ensures fair resource allocation among users by preventing resource monopolization by any single client or IP address.
- **Customizable Policies:** the middleware supports highly configurable rate limiting policies, allowing developers to tailor rules for global, per-client, or per-endpoint scenarios.

- **Response Flexibility:** allows for specifying the HTTP status code returned when the rate limit is exceeded, enabling applications to handle these situations gracefully.

Configuration of the ASP.NET Core Rate Limiting middleware is primarily driven through application settings, typically specified in the `appsettings.json` file. The key aspects of the configuration include:

- **EnableEndpointRateLimiting:** activates rate limiting on individual endpoints.
- **StackBlockedRequests:** determines the behavior for requests exceeding the rate limit (whether they should be queued).
- **RealIpHeader and ClientIdHeader:** designate the headers for identifying the real IP address and client ID, respectively.
- **HttpStatusCode:** the response status code for rate-limited requests.
- **GeneralRules:** specifies global rate limits applicable across all endpoints.
- **ClientRules:** defines rate limits based on the client ID for more granular control.

Developers leveraging DotNetArmor gain access to these functionalities, allowing them to easily implement and manage request rate limiting in their applications. This integration ensures that applications are not only protected against common vulnerabilities but are also equipped to handle excessive traffic volumes gracefully, making DotNetArmor an essential tool for modern web development.

Chapter 8

Machine Learning in Threat Detection: A concrete approach

Machine learning's capability to analyze and learn from data makes it an invaluable asset in identifying cybersecurity threats across various platforms and applications. Beyond databases, ML models are being employed to **detect phishing attempts**, **malware**, **ransomware**, and **anomalous network traffic**, among other cyber threats. These models are trained on vast datasets, allowing them to recognize patterns and anomalies that may elude traditional detection systems.

8.1 Machine Learning tools for enhancing Security

The cybersecurity landscape is rapidly evolving, and machine learning is a key player in bolstering defenses against new threats. Azure Machine Learning stands out with its comprehensive features that simplify building security-focused ML models, making it an essential tool in modern cybersecurity strategies providing benefits and tools like:

- **Phishing Detection**: ML models can analyze email content, URLs, and domain information to identify potential phishing attempts, protecting users from fraudulent sites and emails.
- **Malware and Ransomware Prediction**: by examining file behavior and characteristics, ML can predict whether a file is malicious, even if it's a zero-day malware with no known signature.
- **Network Anomaly Detection**: ML algorithms monitor network traffic in real-time, identifying unusual patterns that could indicate a breach or an ongoing attack.
- **Threat Pattern Recognition in Database Queries**: Azure ML excels in identifying suspicious patterns within database queries, leveraging machine learning to discern between normal operations and potential threats, thus enhancing database security against sophisticated attacks.

In this chapter, I'll discuss how I expanded the DotNetArmor library to include querying an ML-powered REST API, aiming to detect any potential threats to databases.

8.2 Azure Machine Learning

Azure Machine Learning (Azure ML) is a cloud-based service provided by Microsoft, designed to simplify the process of developing, training, and deploying machine learning models. Azure ML offers an end-to-end platform that enables data scientists and developers to build sophisticated machine learning models with efficiency and scalability, without the need to be experts in the algorithms and techniques underlying these models. The platform (or *studio*, as often referred in the documentation), provides several advantages:

- **AutoML**: Azure ML's Automated Machine Learning (AutoML) feature automates the process of selecting the best machine learning algorithms and hyperparameters for your data, significantly reducing the time and expertise required to develop high-performing models.
- **Integration with Other Azure Services**: Azure ML seamlessly integrates with other Azure services, such as Azure Data Lake for storage and Azure Kubernetes Service (AKS) for model deployment, providing a cohesive environment for developing complex, cloud-based machine learning solutions.
- **MLOps Support**: Azure ML includes features that support Machine Learning Operations (MLOps) practices, such as version control, model validation, and deployment pipelines, facilitating the management of machine learning projects at scale.

Despite its goal of simplification, there is still a learning curve associated with understanding how to best utilize Azure ML's features and integrate them into your workflows. Also the cost could represent a potential issue for SMEs and single developers: in fact, costs can vary based on the resources consumed (e.g., compute hours, storage). Hence, it is important for users to manage and monitor their resource usage to control expenses.

8.3 Azure AutoML

As part of the onboarding process for SMEs and users new to the world of Azure ML and database protection, a crucial component is providing an API that can effectively shield against one of the most vicious attacks on databases: SQL injections. These attacks can compromise data security and integrity, making it essential to have the right tools for their prevention.

Developing an API with such capabilities, leveraging machine learning to identify and block attempted SQL injections, demands a deep understanding of both the application domain and advanced machine learning techniques. Designing, training the model, experimenting with various algorithms, selecting the most accurate model, and conducting final testing are all steps that entail a significant time and resource investment, as well as a steep learning curve.

To navigate these challenges and expedite the development of an effective solution, I turned to Azure Machine Learning (Azure ML), particularly its Automated Machine Learning (AutoML) capabilities. Azure AutoML represents a breakthrough in the field of machine learning by automating model selection, hyperparameter tuning, and certain aspects of feature engineering. This not only significantly simplifies

the machine learning model development process but also lowers the entry barrier for less experienced users.

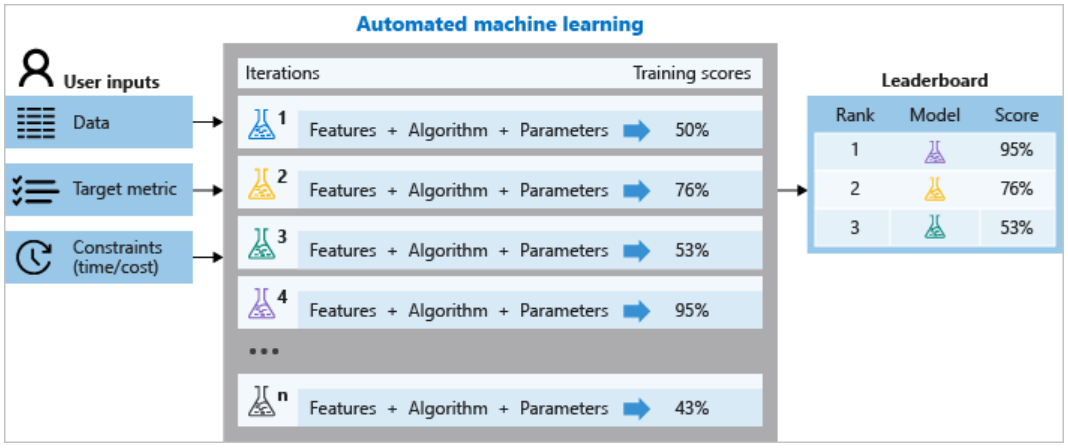


Figure 20: Azure AutoML Schema
Source: `https://learn.microsoft.com`

8.3.1 Cleaning and Preprocessing steps

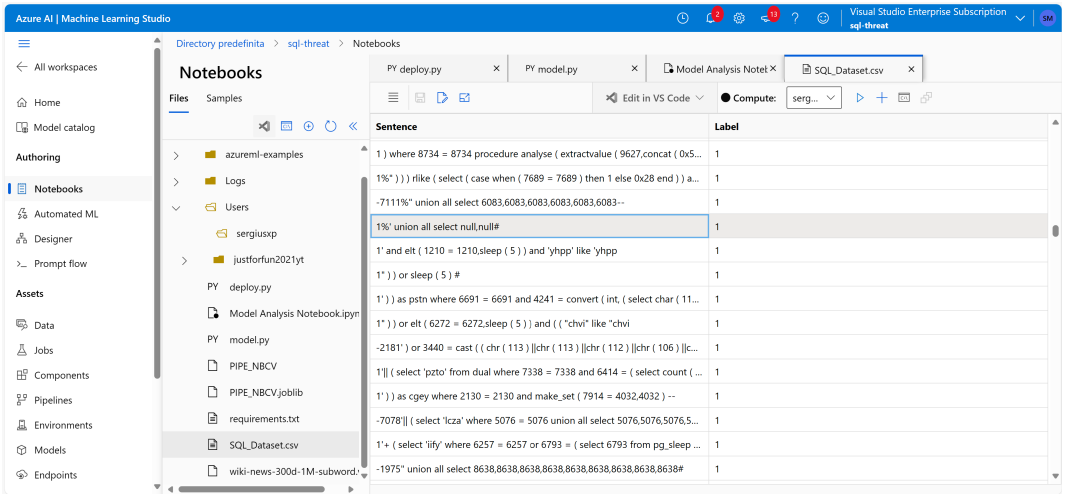
My approach leverages a prepared public dataset, titled `auto_training_sqlinjection_dataset.csv`, composed of SQL sentences alongside corresponding labels — 1 for harmful and 0 for benign queries. It was taken from **Kaggle** [57], a public dataset repository and initially, it encompassed 99,799 samples. However, after a thorough cleaning process aimed at eliminating any instances with null and invalid values, the dataset was refined to 99,527 samples.

The steps included:

- **Renaming**: the column originally labeled `Sentence` was renamed to `SQL` to more accurately reflect its content.
- **Cleaning**: any missing values were dropped to ensure the quality and integrity of the data.
- **Separation**: the dataset was divided into features (SQL queries) and targets (labels), referred to as X and y, respectively.
- **Splitting**: a train-test split was executed, with careful attention to maintaining the stratification of the labels to ensure a balanced representation.

8.3.2 Model Training

Training is a fundamental phase where the model learns to identify and differentiate patterns and characteristics unique to both types of queries. It's during this phase that the model acquires the necessary knowledge to make *informed decisions* about whether a new, unseen query is benign or potentially malicious. Training is essential because it enables the model to **learn from examples**. This learning process involves adjusting the model's internal parameters so that it can accurately



predict the category (benign or malicious) of new SQL queries. Without a comprehensive and well-curated training dataset, the model might struggle to generalize well to new data, leading to inaccurate predictions. The quality and diversity of the training dataset directly influence the model's ability to recognize various forms of SQL injections, making the data gathering phase as critical as the training process itself.

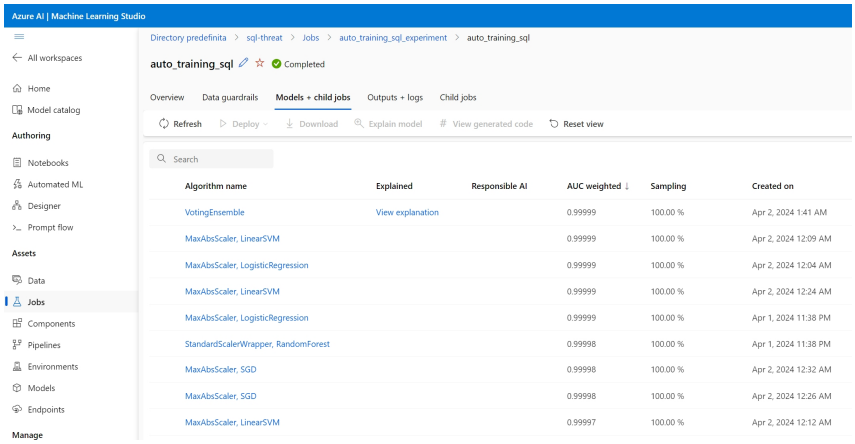
Following an initial preprocessing and cleaning phase conducted via a C# script, I embarked on creating an AutoML experiment within Azure ML. This experiment was powered by an Azure agent, specifically a small instance of type **Standard_F2s_v2** (boasting 2 cores, 4 GB RAM, and 16 GB of disk space), chosen for its cost-efficiency at **0.10 USD per hour of operation**. The training process spanned 2 hours and 14 minutes, during which 48 pre-defined models were evaluated.

8.3.3 ML Algorithms

Selecting the optimal algorithm for a machine learning task is a critical decision that significantly influences the performance and effectiveness of the solution. Even among models with comparable accuracy scores, the choice of algorithm can impact aspects such as interpretability, scalability, and how the model handles new, unseen data. This is especially true in the realm of cybersecurity, where the ability to adapt to evolving threats and maintain high performance over time is paramount. Among the potential candidates identified for enhancing the DotNetArmor library's ability to detect SQL Injection attacks, the following algorithms stood out, each achieving a remarkable AUC of 99.99%:

- **Voting Ensemble**: a powerful meta-algorithm that combines predictions from multiple machine learning models. It leverages the strengths of individual models to improve prediction accuracy and stability.
- **MaxAbsScaler, LinearSVM**: this combination uses MaxAbsScaler for feature scaling, making it compatible with sparse data, and pairs it with a Linear Support Vector Machine (SVM) for classification. SVMs are known for their effectiveness in high-dimensional spaces.

- **MaxAbsScaler, Logistic Regression**: similar to the LinearSVM pairing, this approach scales features using MaxAbsScaler. Logistic Regression, however, offers a probabilistic perspective, making it useful for understanding the likelihood of classifications.



8.3.4 Voting Ensemble

The decision to opt for the Voting Ensemble algorithm was driven by several key considerations:

- **Robustness**: voting Ensemble capitalizes on the collective strengths of various models, inherently providing a more robust solution that is less likely to be swayed by the weaknesses of any single model.
- **Stability and Generalization**: by aggregating predictions, the Voting Ensemble approach tends to offer greater stability and better generalization to new, unseen data. This is crucial for maintaining efficacy against new SQL Injection techniques as attackers evolve.
- **Flexibility**: the ensemble method allows for the inclusion of diverse algorithms, offering flexibility in adapting to the specific characteristics of the SQL Injection detection task.
- **Performance**: despite all five models exhibiting similar accuracy, the Voting Ensemble's ability to leverage multiple learning algorithms provides a safeguard against overfitting, ensuring that the model remains effective across different scenarios and data distributions.

In summary, the choice of the Voting Ensemble algorithm is justified by its superior capacity to integrate diverse predictive models, thereby enhancing the overall prediction accuracy and reliability for SQL Injection detection in the DotNetArmor library. This strategic decision aligns with the goal of offering a cutting-edge, comprehensive security solution that effectively addresses the modern challenges of database protection.

8.3.5 Model Registration

Next step in the creation of the ML-powered API is the model registration, a foundational practice for managing the machine learning lifecycle efficiently. During registration, metadata such as name, description, tags, and model properties can be specified. This step also involves specifying the model's framework (e.g., TensorFlow, PyTorch) and versioning information. It supports:

- **Centralized Management**: provides a single, organized repository for all machine learning models, making management and discovery simpler.
- **Operational Efficiency**: streamlines the deployment process, as registered models can be easily referenced and deployed to various compute targets.
- **Traceability and Auditability**: enhances the traceability of models used in production environments, supporting compliance and audit requirements.
- **Experimentation and Testing**: facilitates A/B testing and experimentation by allowing easy comparison and deployment of different model versions.

After registering the model, Azure ML provides the capability to establish an endpoint based on this model.

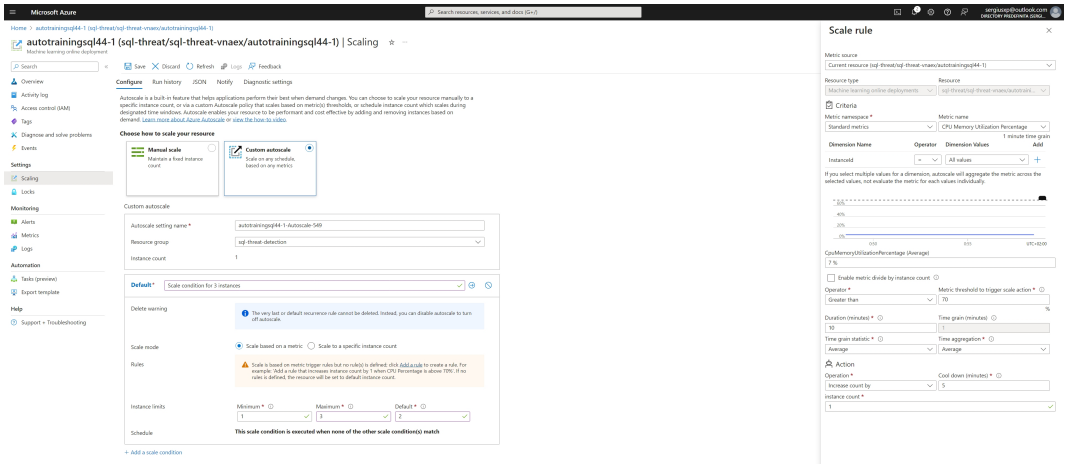
8.3.6 Endpoint Deployment

This process transforms a trained model into a fully functional web service that can receive input data, perform predictions in real-time or batch mode, and return the results. The types of endpoints that can be created are:

- **Real-time Endpoints**: designed for **low-latency**, **real-time predictions**. These endpoints are ideal for applications requiring immediate responses, such as user input validation or interactive websites.
- **Batch Endpoints**: suited for **high-throughput**, **asynchronous processing** of **large datasets**. Batch endpoints are used when the immediate response is not critical, allowing for efficient processing of bulk data in the background.

The deployment utilized a computational agent designated as `Standard_F8s_v2`, equipped with 8 vCPUs, 16 GB RAM, and 64 GB SSD storage. This configuration is recommended by Microsoft Azure specifically for the needs of small and medium enterprises (SMEs). To address potential spikes in demand, a scaling rule was established. This rule is designed to augment the number of instances (up to a maximum of 3 machines) whenever the memory usage exceeds 70%.

Given the particular requirements of the DotNetArmor library, it's crucial to have a REST API that is accessible on-demand, ensuring low latency for swift responses. This need aligns perfectly with the capabilities of real-time endpoints, which are designed to offer immediate predictions and facilitate quick decision-making processes. Furthermore, integrating the real-time endpoint with the DotNetArmor library simplifies the process of updating and maintaining the underlying machine learning models. New versions can be rolled out with minimal downtime, and models can be continuously improved and tested in a production-like environment. This agility is crucial for staying ahead of emerging security threats and adapting to new patterns of SQL injection attacks.



8.4 Improving DB Threat Detection in DotNetArmor

The DotNetArmor library, designed to enhance database security through its C# middleware service, plays a crucial role in assessing SQL predicates for potential threats by leveraging a REST ML API. However, this approach, while effective, introduces **several challenges** that necessitated thoughtful solutions to ensure both efficiency and effectiveness. Below these challenges and the measures taken to address them.

8.4.1 Challenge: Overusage of the REST ML API

One of the primary concerns was the risk of overusing the REST ML API. Excessive calls to the API not only result in increased latency but also higher operational costs and potential rate limiting issues. This is particularly problematic when dealing with a high volume of database queries, as it could slow down application performance and impact user experience.

Improvement: To mitigate this issue, another service was integrated into the DotNetArmor library specifically to regulate API usage. This rate-limiting service ensures that calls to the REST ML API are kept within a sensible threshold, prioritizing critical queries that contain inputs with a higher risk of being malicious. By doing so, the system efficiently balances the need for security with the necessity of maintaining optimal application performance.

8.4.2 Challenge: Overhead from Repeated Queries

Another significant challenge stemmed from the inherent overhead associated with processing the same SQL predicate multiple times. Without an intelligent caching mechanism, identical queries would unnecessarily invoke the REST ML API, wasting resources and increasing response times for threat detection.

Improvement: the introduction of an in-memory cache within the DotNetArmor library addresses this problem directly. By caching the results of SQL predicate evaluations, the system now bypasses the need to re-query the REST ML API for predicates that have already been analyzed. This not only reduces the load on

the API but also significantly decreases the latency involved in threat detection, as cached results can be returned instantaneously.

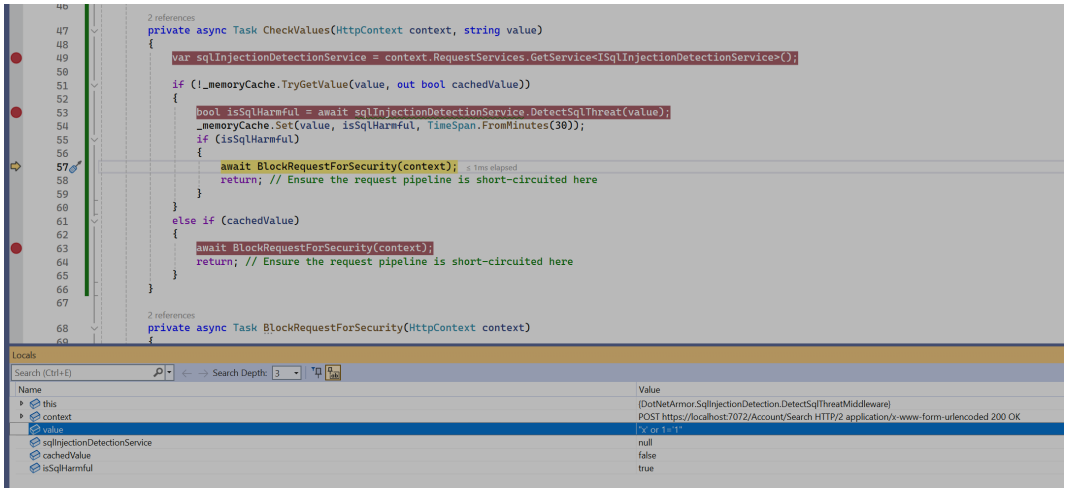
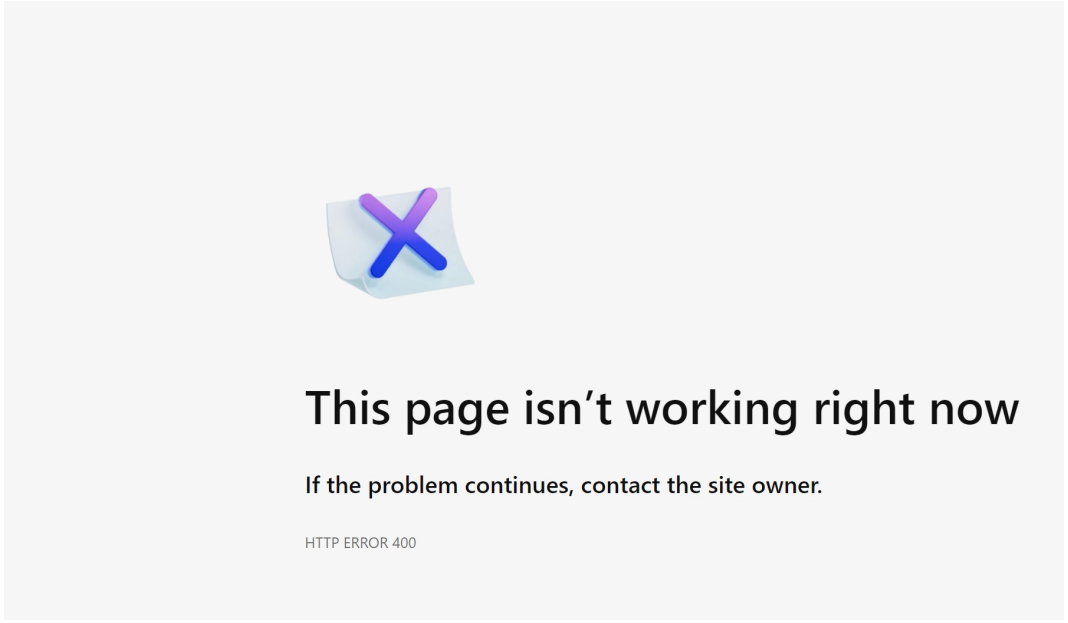


Figure 21: The SQL Threat Detection middleware responds by invoking the ML endpoints and caching their responses in-memory to mitigate excessive and redundant calls. Should a potential threat be identified, the application will issue a status code 400 (Bad Request).



8.4.3 Challenge: Model Accuracy and False Positives/Negatives

Machine learning models, including those used for SQL injection detection, are not infallible. They can produce false positives (benign queries flagged as malicious) and false negatives (malicious queries not detected). High rates of false positives can lead to unnecessary blocking of legitimate database operations, while false negatives might allow harmful queries to slip through.

Mitigation: Continuously monitor and refine the ML model by incorporating new data, including previously misclassified examples. Implement feedback loops where humans can review and correct model predictions, and use this input to retrain the model. Employ ensemble learning techniques or multiple models to improve overall accuracy and reduce the chances of misclassification.

8.4.4 Challenge: Keeping Up with Evolving Threats

Weak Point: Cybersecurity is a constantly changing field, with new types of SQL injection attacks and vulnerabilities emerging regularly. A static ML model might become outdated, losing its effectiveness over time.

Mitigation: Implement a routine for updating the dataset with new types of SQL injections and regularly retrain the model to recognize the latest threats. Adopt techniques like transfer learning to quickly adapt the model to new data. Stay informed about the latest cybersecurity trends and threats to proactively update the system.

8.5 Testing the endpoint

In the development of security-oriented libraries, particularly those aimed at mitigating SQL Injection threats, the effectiveness of such tools must be empirically validated. This validation process involves rigorous testing against a wide array of SQL Injection techniques to ensure that the library not only recognizes and blocks these attempts but also does so without significantly impeding legitimate database queries. The purpose of this is to detail the methodology and importance of testing the Azure Machine Learning endpoint, integrated as a middleware in the library, against a comprehensive set of SQL Injection vectors, including potential false positives. Endpoint testing is crucial for several reasons:

- **Accuracy**: to verify the model's capability in accurately distinguishing between benign and malicious queries.
- **Performance**: to ensure the middleware responds swiftly, minimizing latency in database operations.
- **False Positives/Negatives**: to assess the model's precision in identifying threats without misclassifying legitimate queries as SQL Injection, which could disrupt normal application functionalities.

8.5.1 Testing Methodology

Each SQL Injection vector should be passed to the middleware to evaluate its response. It's critical to:

- Record whether each injection attempt was successfully blocked.
- Monitor for any false positives by submitting legitimate queries that closely mimic SQL Injection patterns.
- Measure the latency introduced by the middleware in processing these queries.

Through systematic testing against a diverse set of SQL Injection vectors and potential false positives, the library's endpoint can be finely tuned for optimal performance and accuracy. This ensures that applications leveraging this library are fortified against SQL Injections, maintaining data integrity and availability without compromising user experience.

8.6 Future Improvements

The ML API, a key feature and service of the DotNetArmor library, allows the detection of database threats through sophisticated machine learning techniques. However, like any advanced system, there is always room for growth and enhancement. Recognizing its current limitations provides a roadmap for future development and improvement.

One notable limitation is the absence of **autonomous learning capabilities** within the machine learning model. Presently, the model does not adapt or learn from new data automatically. This means it cannot independently refine its understanding or improve its accuracy over time without external intervention. Particularly, this limitation manifests in the handling of false positives, incorrect identifications of benign activities as threats. Addressing these inaccuracies requires manual review and adjustment, a process that can be time-consuming and demands a certain level of expertise.

8.6.1 Autolearning and Continuous Improvement

Integrating autolearning capabilities is a pivotal area for future development. By enabling the model to learn from its interactions and feedback continuously, the system can become more accurate and reliable over time. This could involve:

- **Feedback Loops**: implementing mechanisms for users or administrators to flag false positives and negatives. This feedback can be used to adjust the model's parameters or retrain it with updated data, refining its predictive accuracy.
- **Incremental Learning**: developing the model's ability to incorporate new data into its learning process without the need for complete retraining. This approach can make the model more dynamic and responsive to emerging threats.
- **Automated Retraining Pipelines**: establishing automated processes that periodically retrain the model with the latest data, including newly identified threats and feedback on its performance. This ensures the model remains up-to-date with the evolving landscape of database threats.

8.6.2 Enhancing Model Retraining

The necessity for manual intervention in model retraining is a critical area for improvement. Future versions of the DotNetArmor ML API could benefit from:

- **Semi-Automatic Retraining**: while complete automation of retraining might be challenging, introducing semi-automatic triggers based on certain criteria (e.g., a threshold of identified false positives) could streamline the process.

- **Streamlined Data Annotation**: simplifying the process for labeling data, particularly for non-experts, can enhance the model's training and retraining processes. This could involve user-friendly interfaces for data annotation and feedback submission.
- **Advanced Anomaly Detection**: incorporating more sophisticated anomaly detection algorithms to better distinguish between genuine threats and false alarms. This could reduce the reliance on manual review by improving the initial accuracy of threat identification.

In conclusion, while the ML API within the library has made significant difference in database threat detection, embracing the path towards autolearning and minimizing manual intervention in model retraining presents a promising avenue for elevating its capabilities. These future improvements not only aim to enhance the system's efficiency and accuracy but also ensure that it remains resilient against the rapidly evolving threats in cybersecurity.

Source Code

Appendix A

Source Code

DotNetArmor is a .NET class library designed to assist developers and SMEs by providing a ready-to-use security boilerplate. This includes third-party libraries, security algorithms optimized for the library, bespoke new algorithms, and database threat detection APIs. Consequently, the library is logically divided into macro services based on utility (Cryptography, Passwords, Hashing, Database, etc.). For ease of use, it was decided to access these services through Dependency Injection, a design pattern in programming that makes applications more flexible, testable, and modular by reducing coupling between classes and managing dependencies in a centralized manner.

A.1 Service Registration

Upon completion, the library is compiled through a CLI (Command Line Interface) or an IDE (Integrated Development Environment), producing a DLL (Dynamic Link Library) that can be consumed by .NET projects.

As highlighted in the introductory section, the library encompasses a suite of tools, which comprises both third-party libraries and services necessitating API keys, for instance, the *Have I Been Pwned?* service, *KeyVault*, or database functionalities. While incorporating this library into a project, it's important to note that utilizing all these components is optional. However, for the comprehensive development of the library, it is imperative to register each component:

```
1  // Adding and configuring DotNetArmor for KeyVault
2  builder.Services.Configure<KeyVaultConfig>(
3          builder.Configuration.GetSection("AzureKeyVault"));
4  builder.Services.AddSingleton<IAzureKeyVaultService,
       AzureKeyVaultService>();
5  // Configuring DotNetArmor for Databases
6  builder.Services.Configure<SqlServiceOptions>(
7          builder.Configuration.GetSection("SqlService"));
8  builder.Services.AddSingleton<ISqlService, SqlService>();
9  // Configuring SqlInjection Endpoint
10 builder.Services.Configure<SqlThreatDetectionOptions>(
11         builder.Configuration.GetSection("SqlInjectionEndpointSecret"));
12 builder.Services.AddSingleton<ISqlInjectionDetectionService,
       SqlInjectionDetectionService>();
13 // Configuring DotNetArmor for HIBP Service
```

```
builder.Services.Configure<PasswordServiceOptions>(
        builder.Configuration.GetSection("HibpApiKey"));
// Registering Rate API Limit
builder.Services.Configure<IpRateLimitOptions>(
        builder.Configuration.GetSection("IpRateLimiting"));
builder.Services.Configure<IpRateLimitPolicies>(
        builder.Configuration.GetSection("IpRateLimitPolicies"));
// Registering DotNetArmor Services
builder.Services.AddDotNetArmorLibrary();
```

This way, our application can inject various configurations and keys that will be used in our services.

A.2 Services

A.2.1 AzureKeyVaultService

```
public interface IAzureKeyVaultService
{
    public Task<string> GetSecretAsync(string secretName);
    public Task SetSecretAsync(string secretName, string value);
    public Task DeleteSecretAsync(string secretName);

}

public class AzureKeyVaultService : IAzureKeyVaultService
{
    private readonly SecretClient _secretClient;

    public AzureKeyVaultService(IOptions<KeyVaultConfig> config)
    {
        var clientSecretCredential = new
            ClientSecretCredential(config.Value.TenantId,
            config.Value.ClientId, config.Value.ClientSecret);
        _secretClient = new SecretClient(new
            Uri(config.Value.KeyVaultUrl), clientSecretCredential);
    }

    public async Task<string> GetSecretAsync(string secretName)
    {
        KeyVaultSecret secret = await
            _secretClient.GetSecretAsync(secretName);
        return secret.Value;
    }

    public async Task SetSecretAsync(string secretName, string value)
    {
        await _secretClient.SetSecretAsync(new
            KeyVaultSecret(secretName, value));
    }

    public async Task DeleteSecretAsync(string secretName)
    {
        await _secretClient.StartDeleteSecretAsync(secretName);
    }
}
```

A.2.2 SQLInjectionDetectionService

```
1  public interface ISqlInjectionDetectionService
2  {
3      public Task<bool> DetectSqlThreat(string predicate);
4  }
5
6  public class SqlInjectionDetectionService : ISqlInjectionDetectionService
7  {
8      private readonly string _sqlTreatEndpointKey;
9
10     public
           SqlInjectionDetectionService(IOptions<SqlThreatDetectionOptions>
           config)
11     {
12         _sqlTreatEndpointKey = config.Value.SqlThreatDetectionKey;
13     }
14
15     public async Task<bool> DetectSqlThreat(string predicate)
16     {
17         var handler = new HttpClientHandler()
18         {
19             ClientCertificateOptions = ClientCertificateOption.Manual,
20             ServerCertificateCustomValidationCallback =
21                     (httpRequestMessage, cert, cetChain, policyErrors)
                           => { return true; }
22         };
23
24         using (var client = new HttpClient(handler))
25         {
26             if (string.IsNullOrEmpty(_sqlTreatEndpointKey))
27             {
28                 throw new Exception("A key should be provided to invoke
                       the endpoint");
29             }
30             client.DefaultRequestHeaders.Authorization = new
                   AuthenticationHeaderValue("Bearer", _sqlTreatEndpointKey);
31             client.BaseAddress = new
                   Uri("https://sql-threat-vnaex.westeurope.inference.ml.azure.com/score");
32
33             SqlThreatModel model = new SqlThreatModel();
34             model.InputData.Columns = new List<string>() { "Sentence" };
35             model.InputData.Index = new List<object>() { 0 };
36             model.InputData.Data = new List<object>() { predicate };
37             var content = new
                   StringContent(JsonSerializer.Serialize(model));
38             content.Headers.ContentType = new
                   MediaTypeHeaderValue("application/json");
39
40             content.Headers.Add("azureml-model-deployment",
                   "autotrainingsql44-1");
41             HttpResponseMessage response = await client.PostAsync("",
                   content);
42
43             if (response.IsSuccessStatusCode)
44             {
45                 string result = await
                       response.Content.ReadAsStringAsync();
46                 return result == "[1]";
47             }
48             else
```

```
49              {
50                  Console.WriteLine(string.Format("The request failed with
                        status code: {0}", response.StatusCode));
51                  Console.WriteLine(response.Headers.ToString());
52                  string responseContent = await
                        response.Content.ReadAsStringAsync();
53                  Console.WriteLine(responseContent);
54
55                  return false;
56              }
57          }
58      }
59  }
```

A.2.3 DetectSqlThreatMiddleware

```
1   [AttributeUsage(AttributeTargets.Class | AttributeTargets.Method)]
2   public class DetectSqlThreat : Attribute { }
3   public class DetectSqlThreatMiddleware
4   {
5       private readonly RequestDelegate _next;
6       private readonly IMemoryCache _memoryCache;
7
8       public DetectSqlThreatMiddleware(
9           RequestDelegate next,
10          IMemoryCache memoryCache)
11      {
12          _next = next;
13          _memoryCache = memoryCache;
14      }
15
16      public async Task Invoke(HttpContext context)
17      {
18          if
                (context.GetEndpoint()?.Metadata.GetMetadata<DetectSqlThreat>()
                != null)
19          {
20              foreach (string key in context.Request.Query.Keys)
21              {
22                  var value = context.Request.Query[key];
23                  await CheckValues(context, value);
24              }
25
26              foreach (var key in context.Request.Form.Keys)
27              {
28                  var value = context.Request.Form[key];
29
30                  if (value != "__RequestVerificationToken")
31                  {
32                      await CheckValues(context, value);
33                  }
34              }
35          }
36
37          await _next(context);
38      }
39
40      private async Task CheckValues(HttpContext context, string value)
41      {
```

```
42          var sqlInjectionDetectionService =
                context.RequestServices.GetService<ISqlInjectionDetectionService>();
43
44          if (!_memoryCache.TryGetValue(value, out bool cachedValue))
45          {
46              bool isSqlHarmful = await
                    sqlInjectionDetectionService.DetectSqlThreat(value);
47              _memoryCache.Set(value, isSqlHarmful,
                    TimeSpan.FromMinutes(30));
48              if (isSqlHarmful)
49              {
50                  await BlockRequestForSecurity(context);
51                  return; // Ensure the request pipeline is
                        short-circuited here
52              }
53          }
54          else if (cachedValue)
55          {
56              await BlockRequestForSecurity(context);
57              return; // Ensure the request pipeline is short-circuited
                    here
58          }
59      }
60
61      private async Task BlockRequestForSecurity(HttpContext context)
62      {
63          context.Response.StatusCode = (int)HttpStatusCode.BadRequest;
64          await context.Response.WriteAsync(JsonSerializer.Serialize(new
65          {
66              Msg = "Potentially dangerous query blocked for security
                    reasons."
67          }));
68          await context.Response.CompleteAsync();
69      }
70  }
71
72  public static class MiddlewareExtensions
73  {
74      public static IApplicationBuilder UseSqlThreatMiddleware(this
            IApplicationBuilder builder)
75      {
76          return builder.UseMiddleware<DetectSqlThreatMiddleware>();
77      }
78  }
```

A.2.4 PasswordService and *Have I Been Pwned?* Service

```
1   public interface IPasswordService
2   {
3       public string GenerateSecurePassword(int length);
4       public string GenerateSalt(int size);
5       public Task<PwnedEmailServiceModel[]?> IsEmailPwnedAsync(string
            email);
6       public Task<bool> IsPasswordPwnedAsync(string password);
7       public string CheckStrength(string password);
8   }
9
10  public class PasswordService : IPasswordService
11  {
```

```
12      private readonly string _hibpApiKey;
13      private readonly HttpClient _httpClient;
14      private readonly IHashingService _hashingService;
15
16      public PasswordService(
17          IOptions<PasswordServiceOptions> options,
18          HttpClient httpClient,
19          IHashingService hashingService)
20      {
21          _hibpApiKey = options.Value.Key;
22          _httpClient = httpClient;
23          _httpClient.DefaultRequestHeaders.Add("User-Agent", "HIBP
                Checker");
24          _hashingService = hashingService;
25      }
26
27      public string CheckStrength(string password)
28      {
29          int score = 0;
30
31          if (string.IsNullOrWhiteSpace(password))
32              return "Password is empty";
33
34          // Length check
35          if (password.Length < 8)
36              return "Very Weak"; // Length is less than 8 characters
37          else
38              score++;
39
40          if (password.Any(char.IsLower))
41              score++; // Increment if password has lowercase letters
42
43          if (password.Any(char.IsUpper))
44              score++; // Increment if password has uppercase letters
45
46          if (password.Any(char.IsDigit))
47              score++; // Increment if password has digits
48
49          if (password.Any(ch => !char.IsLetterOrDigit(ch)))
50              score++; // Increment if password has special characters
51
52          return score switch
53          {
54              1 => "Very Weak",
55              2 => "Weak",
56              3 => "Medium",
57              4 => "Strong",
58              5 => "Very Strong",
59              _ => "Unknown",
60          };
61      }
62
63      public async Task<bool> IsPasswordPwnedAsync(string password)
64      {
65          if (_hibpApiKey == "")
66          {
67              string link = "https://haveibeenpwned.com/API/Key";
68              throw new Exception($"Please, provide a valid HIBP API Key.
                    To retrieve one, please visit {link}.");
69          }
70
```

```
71          string hashString =
                _hashingService.ComputeSha1(password).ToUpper();
72
73          string apiEndpoint =
                $"https://api.pwnedpasswords.com/range/{hashString.Substring(0,
                5)}";
74          var response = await _httpClient.GetAsync(apiEndpoint);
75          string content = await response.Content.ReadAsStringAsync();
76          if (!response.IsSuccessStatusCode)
77          {
78              throw new Exception($"Error when calling a HIBP Api:
                    ${content}");
79          }
80
81          return content.Contains(hashString[5..]);
82      }
83
84      public async Task<PwnedEmailServiceModel[]?>
            IsEmailPwnedAsync(string email)
85      {
86          if (_hibpApiKey == "")
87          {
88              string link = "https://haveibeenpwned.com/API/Key";
89              throw new Exception($"Please, provide a valid HIBP API Key.
                    To retrieve one, please visit {link}.");
90          }
91
92          _httpClient.DefaultRequestHeaders.Add("hibp-api-key",
                _hibpApiKey);
93
94          string apiEndpoint =
                $"https://haveibeenpwned.com/api/v3/breachedaccount/{email}";
95          var response = await _httpClient.GetAsync(apiEndpoint);
96          string content = await response.Content.ReadAsStringAsync();
97
98          if (response.StatusCode == System.Net.HttpStatusCode.NotFound)
99          {
100             return [];
101         }
102
103         if (!response.IsSuccessStatusCode)
104         {
105             throw new Exception($"Error when calling a HIBP Api:
                    ${content}");
106         }
107
108         return
                JsonSerializer.Deserialize<PwnedEmailServiceModel[]?>(content);
109     }
110
111     public string GenerateSecurePassword(int length = 16)
112     {
113         const string validChars =
                "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890!@#$%^&*()";
114         var byteArr = new byte[length];
115
116         using (var rng = RandomNumberGenerator.Create())
117         {
118             rng.GetBytes(byteArr);
119         }
120
```

```
121         char[] chars = new char[length];
122         for (int i = 0; i < length; i++)
123         {
124             chars[i] = validChars[byteArr[i] % validChars.Length];
125         }
126 
127         return new string(chars);
128     }
129 
130     public string GenerateSalt(int size)
131     {
132         var saltBytes = new byte[size];
133 
134         using (var rng = RandomNumberGenerator.Create())
135         {
136             rng.GetBytes(saltBytes);
137         }
138 
139         return Convert.ToBase64String(saltBytes);
140     }
141 }
```

A.2.5 SqlService

```
1  public interface ISqlService
2  {
3      public DataTable ExecuteQuery(string commandText, CommandType
           commandType, params SqlParameter[] parameters);
4      public DataTable ExecuteNonParametrizedQuery(string commandText,
           CommandType commandType);
5  }
6  
7  public class SqlService : ISqlService
8  {
9      private readonly string _connectionString;
10 
11     public SqlService(IOptions<SqlServiceOptions> options)
12     {
13         _connectionString = options.Value.ConnectionString;
14     }
15 
16     public DataTable ExecuteQuery(string commandText, CommandType
           commandType, params SqlParameter[] parameters)
17     {
18         using (var connection = new SqlConnection(_connectionString))
19         {
20             connection.Open();
21             using (var command = new SqlCommand(commandText, connection))
22             {
23                 command.CommandType = commandType;
24                 command.Parameters.AddRange(parameters);
25                 using (var adapter = new SqlDataAdapter(command))
26                 {
27                     var dataTable = new DataTable();
28                     adapter.Fill(dataTable);
29                     return dataTable;
30                 }
31             }
32         }
```

```
33      }
34
35      public DataTable ExecuteNonParametrizedQuery(string commandText,
            CommandType commandType)
36      {
37          using (var connection = new SqlConnection(_connectionString))
38          {
39              connection.Open();
40              using (var command = new SqlCommand(commandText, connection))
41              {
42                  command.CommandType = commandType;
43                  using (var adapter = new SqlDataAdapter(command))
44                  {
45                      var dataTable = new DataTable();
46                      adapter.Fill(dataTable);
47                      return dataTable;
48                  }
49              }
50          }
51      }
52  }
```

A.2.6 Input validation example: Password Strength Attribute

```
1   public class PasswordStrengthAttribute : ValidationAttribute
2   {
3       public string[] AcceptableStrengths { get; set; }
4
5       public PasswordStrengthAttribute(params string[] strengths)
6       {
7           AcceptableStrengths = strengths;
8       }
9
10      protected override ValidationResult IsValid(object value,
            ValidationContext validationContext)
11      {
12          // Access the DI container to resolve PasswordService
13          var serviceProvider =
                validationContext.GetService(typeof(IPasswordService));
14          var passwordService = serviceProvider as IPasswordService;
15
16          if (passwordService == null)
17          {
18              throw new InvalidOperationException("PasswordService not
                    found.");
19          }
20
21          if (value == null || string.IsNullOrWhiteSpace(value.ToString()))
22          {
23              return new ValidationResult("Password is required.");
24          }
25
26          var strength = passwordService.CheckStrength(value.ToString());
27
28          if (AcceptableStrengths.Contains(strength))
29          {
30              return ValidationResult.Success;
31          }
32
```

```
33          var acceptable = string.Join(", ", AcceptableStrengths);
34          return new ValidationResult($"The password does not meet the
                required strength. Acceptable strengths are: {acceptable}.");
35      }
36  }
```

Bibliography

[1] M. Aslett (451 Research) A. Pavlo (Carnegie Mellon University). Newsql, what is? https://db.cs.cmu.edu/papers/2016/pavlo-newsql-sigmodrec2016.pdf, 2016.

[2] Database Of Databases authors. Cosmos db's history. https://dbdb.io/db/cosmos-db, 2022.

[3] Polly Authors. Polly library. https://github.com/App-vNext/Polly, 2024.

[4] S. Sanfilippo (Blog). Salvatore sanfilippo's personal blog. http://www.invece.org/, 2024.

[5] H. Bar (Bright). SQL Injection Attack - BrightSec. https://brightsec.com/blog/sql-injection-attack/, 2023.

[6] G. D. Maayan (CloudTweaks). 5 azure cost management strategies. https://cloudtweaks.com/2024/03/5-azure-cost-management-strategies/, 2024.

[7] C. Sanders (Microsoft Corp.). Microsoft cloud for sovereignty: The most flexible and comprehensive solution for digital sovereignty. https://blogs.microsoft.com/blog/2022/07/19/microsoft-cloud-for-sovereignty-the-most-flexible-and-comprehensive-solu 2022.

[8] E. F. Codd (IBM Corp.). Introduction of the relational database model. https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf, 1970.

[9] Gustavo M. (Microsoft Corp.). Intrusion Detection and Prevention System based on Azure - Microsoft Tech Community. https://techcommunity.microsoft.com/t5/azure-network-security-blog/intrusion-detection-and-prevention-system-idps-based-on/ba-p/3921330, 2023.

[10] IBM Corp. Relational database systems. https://www.ibm.com/topics/relational-databases, 2022.

[11] Microsoft Corp. Microsoft at postgresopen 2017. https://azure.microsoft.com/es-es/blog/microsoft-at-postgresopen-2017/, 2017.

[12] Microsoft Corp. What is iaas? https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-iaas, 2021.

[13] Microsoft Corp. What is a virtual machine? https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-a-virtual-machine, 2022.

[14] Microsoft Corp. Azure blueprints overview. https://learn.microsoft.com/en-us/azure/governance/blueprints/overview, 2023.

[15] Microsoft Corp. Azure management groups overview. https://learn.microsoft.com/en-us/azure/governance/management-groups/overview, 2023.

[16] Microsoft Corp. Azure policy overview. https://learn.microsoft.com/en-us/azure/governance/policy/overview, 2023.

[17] Microsoft Corp. Sql azure vulnerability assessment. https://learn.microsoft.com/en-us/azure/defender-for-cloud/sql-azure-vulnerability-assessment-overview, 2023.

[18] Microsoft Corp. Encryption of data in transit. https://learn.microsoft.com/en-us/azure/security/fundamentals/encryption-overview, 2023.

[19] Microsoft Corp. Dynamic data masking overview. https://learn.microsoft.com/en-us/azure/azure-sql/database/dynamic-data-masking-overview?view=azuresql, 2023.

[20] Microsoft Corp. Data classification in azure. https://learn.microsoft.com/en-us/azure/well-architected/security/data-classification, 2023.

[21] Microsoft Corp. Secure least privileged access. https://learn.microsoft.com/en-us/entra/identity-platform/secure-least-privileged-access, 2023.

[22] Microsoft Corp. Azure sql security baseline. https://learn.microsoft.com/en-us/security/benchmark/azure/baselines/azure-sql-security-baseline, 2023.

[23] Microsoft Corp. How Multi-Factor Authentication Works - Microsoft Entra Docs. https://learn.microsoft.com/en-us/entra/identity/authentication/concept-mfa-howitworks, 2023.

[24] Microsoft Corp. Multi-Factor Authentication for Azure SQL Database - Microsoft Docs. https://learn.microsoft.com/en-us/azure/azure-sql/database/authentication-mfa-ssms-overview?view=azuresql, 2023.

[25] Microsoft Corp. Conditional Access in Azure Active Directory - Microsoft Docs. https://learn.microsoft.com/en-us/entra/identity/conditional-access/overview, 2023.

[26] Microsoft Corp. Azure SQL Database Threat Detection - Microsoft Docs. https://learn.microsoft.com/en-us/azure/azure-sql/database/threat-detection-overview?view=azuresql, 2023.

[27] Microsoft Corp. Azure compliance manager. https://learn.microsoft.com/it-it/purview/compliance-manager, 2024.

[28] Microsoft Corp. Azure cost management and billing overview. https://learn.microsoft.com/en-us/azure/cost-management-billing/cost-management-billing-overview, 2024.

[29] Microsoft Corp. Azure cli. https://learn.microsoft.com/en-us/cli/azure/, 2024.

[30] Microsoft Corp. Azure storage sdks. https://learn.microsoft.com/en-us/dotnet/api/overview/azure/storage?view=azure-dotnet, 2024.

[31] Microsoft Corp. Azure security principles. https://learn.microsoft.com/en-us/azure/well-architected/security/principles, 2024.

[32] Microsoft Corp. Data minimization in azure. https://learn.microsoft.com/en-us/privacy/priva/risk-management-policy-data-minimization, 2024.

[33] Microsoft Corp. Azure policy. https://azure.microsoft.com/en-us/products/azure-policy, 2024.

[34] Microsoft Corp. Gdpr data subject requests in office 365. https://learn.microsoft.com/en-us/compliance/regulatory/gdpr-dsr-office365, 2024.

[35] Microsoft Corp. Azure sql deployment options. https://learn.microsoft.com/en-us/training/modules/azure-sql-intro/3-deployment-options, 2024.

[36] Microsoft Corp. Azure defender for cloud. https://azure.microsoft.com/en-us/products/defender-for-cloud/, 2024.

[37] Microsoft Corp. Transparent data encryption overview. https://learn.microsoft.com/it-it/azure/azure-sql/database/transparent-data-encryption-tde-overview, 2024.

[38] Microsoft Corp. Azure sql database authentication. https://learn.microsoft.com/en-us/azure/azure-sql/database/authentication-aad-configure?view=azuresql, 2024.

[39] Microsoft Corp. Monitoring azure sql database with azure monitor. https://learn.microsoft.com/en-us/azure/azure-sql/database/monitoring-sql-database-azure-monitor?view=azuresql, 2024.

[40] Microsoft Corp. Backup and restore in azure sql. https://learn.microsoft.com/en-us/training/modules/backup-restore-azure-sql/, 2024.

[41] Microsoft Corp. Azure Firewall Overview - Microsoft Docs. https://learn.microsoft.com/en-us/azure/firewall/overview, 2024.

[42] Microsoft Corp. Configure Azure SQL Database Firewall - Microsoft Docs. https://learn.microsoft.com/en-us/azure/azure-sql/database/firewall-configure, 2024.

[43] Microsoft Corp. Microsoft Azure Portal. https://portal.azure.com, 2024.

[44] Microsoft Corp. Azure Role-Based Access Control Built-in Roles - Microsoft Docs. https://learn.microsoft.com/en-us/azure/role-based-access-control/built-in-roles, 2024.

[45] Microsoft Corp. What's happening to mariadb in azure. https://learn.microsoft.com/en-us/azure/mariadb/whats-happening-to-mariadb, 2024.

[46] Microsoft Corp. Azure portal. https://portal.azure.com, 2024.

[47] N. Maheshwary (Microsoft Corp.). Tokenization with external tokenization systems in azure. https://techcommunity.microsoft.com/t5/azure-architecture-blog/data-analytics-solution-tokenization-with-external-tokenization/ba-p/2821038, 2022.

[48] (R. Abandy) Microsoft Corp. The history of microsoft azure. https://techcommunity.microsoft.com/t5/educator-developer-blog/the-history-of-microsoft-azure/ba-p/3574204, 2022.

[49] S. Cuff (Microsoft Corp.). What's the difference between azure security center, azure defender, and azure sentinel? https://techcommunity.microsoft.com/t5/itops-talk-blog/what-s-the-difference-between-azure-security-center-azure/ba-p/2155188, 2021.

[50] N. Srav et al. (Owasp). Denial of Service - OWASP. https://owasp.org/www-community/attacks/Denial_of_Service, 2021.

[51] European Parliament (Europe). Gdpr article 5. https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679, 2016.

[52] Emil S. (Ntirety for Forbes). How cloud computing revolutionized business operations and what lies ahead. https://www.forbes.com/sites/emilsayegh/2023/11/28/how-cloud-computing-revolutionized-business-operations-and-what-lies-ahe
2023.

[53] MAPL for LinkedIn. The future of cloud computing: 2025-2030. https://www.linkedin.com/pulse/future-cloud-computing-2025-2030-maplworld, 2023.

[54] R. David (Hyas). Proactive Intelligence: A Paradigm Shift in Cyber Defense - HYAS. https://www.hyas.com/blog/

proactive-intelligence-a-paradigm-shift-in-cyber-defense, 2024.

[55] Washington University in St. Louis. Cia triad. https://informationsecurity.wustl.edu/items/confidentiality-integrity-and-availability-the-cia-triad/, 2024.

[56] Oracle Inc. Sql as relational dbms standard. https://docs.oracle.com/cd/B13789_01/server.101/b10759/intro001.htm, 2022.

[57] Kaggle.com. Kaggle's free sql injection dataset. https://www.kaggle.com/datasets/sajid576/sql-injection-dataset, 2024.

[58] E. Jones (Kinsta). Cloud market share – a look at the cloud ecosystem in 2023. https://kinsta.com/blog/cloud-market-share/, 2024.

[59] R. H. Sheikh (KnowledgeHut). What is rdbms? https://www.knowledgehut.com/blog/database/what-is-rdbms, 2022.

[60] K. Niewiarowska (LearnSQL). History of ms sql server. https://learnsql.com/blog/history-ms-sql-server/, 2021.

[61] Microsoft. Antixss library. https://learn.microsoft.com/it-it/dotnet/api/system.web.security.antixss?view=netframework-4.8.1, 2024.

[62] Microsoft. Microsoft azure keyvault service. https://azure.microsoft.com/en-us/products/key-vault, 2024.

[63] MongoDB. Acid. https://www.mongodb.com/basics/acid-transactions, 2022.

[64] Government of California (United States of America). California consumer privacy act of 2018. https://leginfo.legislature.ca.gov/faces/codes_displayText.xhtml?division=3.&part=4.&lawCode=CIV&title=1.81.5, 2018.

[65] A. L. García (Massachuset Institute of Technology). A brief history of cloud computing. https://dspace.mit.edu/bitstream/handle/1721.1/100311/932065967-MIT.pdf;sequence=1, 2022.

[66] M. Rick (Owasp). SQL Injection - OWASP. https://owasp.org/www-community/attacks/SQL_Injection, 2022.

[67] Quickbase. Timeline of database history. https://www.quickbase.com/articles/timeline-of-database-history, 2022.

[68] E. Everson (Cleveland State Law University). Privacy by design: Origin and purpose. https://engagedscholarship.csuohio.edu/cgi/viewcontent.cgi?article=3933&context=clevstlrev, 2022.

[69] M. Buckbee (Varonis). Azure compliance. https://www.varonis.com/blog/azure-compliance, 2022.

[70] Brachiosoft with elements of interviews to Salvatore Sanfilippo. Redis' history. https://blog.brachiosoft.com/en/posts/redis/, 2023.