First published in 2026 by UE Campus

Unit specifications can be found on the UE Campus Portal: https://uecampus.com/

Contents

Using your Study Guide

Welcome to the study guide, designed to support you in completing your Level 4 Diploma in Cyber Security.

This study guide follows the order of the syllabus, which is the basis for your studies. Each chapter starts by listing the syllabus learning outcomes covered and the assessment criteria.

Level 4 Units

The Level 4 Diploma in Cyber Security consists of the following units:

Unit Title	Credits	Status
Cyber Security Threat and Risk	20	Mandatory
Network Security and Data Communications	20	Mandatory
Database Security and Computer Programming	20	Mandatory
Incident Response, Investigations and Forensics	20	Mandatory
Security Strategy: Laws, Policies and Implementation	20	Mandatory
Cyber Security Threats and Risk: Banking and Finance	20	Optional
Cyber Wars	20	Optional

The study guide includes a number of features to enhance your studies:

📝	**'Over to you:'** activities for you to apply what you have learned.
🌎	**'Industry Insights:'** discover up-to-date trends, expert opinions, and real-world examples from leading organisations.
🧭	**'Did you know?'** highlights interesting facts or surprising information to deepen your understanding.
💼	**'Case studies:'** realistic business scenarios to reinforce and test your understanding.
❗	**'Need to know:'** key pieces of information highlighted in the text.
💡	**'Examples:'** illustrating points made in the text to show how it works in practice.

Note: Website addresses current as of March 2026.

Level 4 Database Security and Computer Programming

About this unit

Database security concerns the use of a broad range of information security controls to protect databases – potentially including the data itself, the database applications or stored functions, the database systems, the database servers, and the associated network links – against compromises of their confidentiality, integrity, and availability.

In this unit you will explore security risks to database systems and the mitigation techniques used to protect them. You will examine the different types of databases, how they organise data, and the categories of control that can be applied to secure them. You will also investigate cloud-based storage solutions and the tools available to data owners, custodians, incident responders, and investigators.

The second half of this unit explores the fundamental relationship between computer programming and computer hacking. Understanding how software is written is essential to understanding how it can be exploited. You will examine popular programming languages, analyse the symbiotic link between programming skills and hacking capabilities, and investigate Python as a case study – examining how this versatile language is used by both security professionals and malicious actors.

By the end of this unit, you will be able to explain database security risks, assess the effectiveness of database protection tools, understand cloud-based storage security, analyse the relationship between programming and hacking, and investigate how Python is used in the cyber security landscape.

Unit code: **M/617/1131**

RQF level: **4**

Credits: **20**

Assessment: **Written Assignment – Database Security and Programming Analysis Report**

Chapter One – Information Security Controls for Database Protection

Introduction

This chapter examines the security risks facing database systems and evaluates the information security concepts and tools used to protect them. Databases are among the most valuable targets for cyber attackers because they store concentrated collections of sensitive data – from customer personal information and financial records to intellectual property and trade secrets. A single database breach can expose millions of records and cause catastrophic financial, reputational, and legal damage.

Understanding database security requires knowledge of how databases work, the specific threats they face, and the technical, procedural, and physical controls available to mitigate those threats. This chapter provides that foundation.

Learning Outcomes

On completing the chapter, you will be able to:

1. **Understand the broad range of information security controls to protect databases.**

Assessment Criteria

1.1 Explain security risks in database systems.

1.2 Assess the effectiveness of information security concepts and tools in protecting databases.

1.1 Security risks in database systems

Over to you – Video Watch: Database Fundamentals

Watch this YouTube video:

Title: What is a Database? – Lucidchart

Duration: 9:51

Link: https://www.youtube.com/watch?v=wR0jg0eQsZA&t=13s

After watching, explain in your own words: (a) what a database is, (b) the difference between a relational and non-relational database, and (c) why databases are attractive targets for attackers.

Types of Databases and How They Organise Data

A database is an organised collection of structured data, typically stored and accessed electronically from a computer system. Understanding the different types of databases is essential for understanding their security requirements:

- **Relational databases (RDBMS)** – organise data into tables (relations) with rows and columns. Tables are linked through keys (primary keys and foreign keys). Examples include MySQL, PostgreSQL, Microsoft SQL Server, and Oracle Database.

Relational databases use Structured Query Language (SQL) for data manipulation and are the most widely deployed database type in enterprise environments.

- **NoSQL databases** – designed for specific data models with flexible schemas. Types include document databases (MongoDB, CouchDB), key-value stores (Redis, DynamoDB), wide-column stores (Cassandra, HBase), and graph databases (Neo4j). NoSQL databases are increasingly popular for big data, real-time applications, and cloud-native architectures.
- **Object-oriented databases** – store data as objects, similar to object-oriented programming. Used in applications requiring complex data relationships, such as CAD/CAM systems and multimedia applications.
- **In-memory databases** – store data in main memory (RAM) rather than on disk, providing extremely fast read and write operations. Used for real-time analytics, caching, and session management. Examples include Redis and SAP HANA.
- **Cloud databases** – hosted on cloud platforms (AWS RDS, Azure SQL Database, Google Cloud SQL) and managed as a service. Cloud databases introduce additional security considerations around shared responsibility, data sovereignty, and multi-tenancy.

Database Security Breach Types

Database systems face a wide range of security threats. The principal breach types include:

- **SQL injection** – the most prevalent database attack. Attackers insert malicious SQL statements through application input fields to manipulate the database – extracting data, modifying records, deleting tables, or gaining administrative access. SQL injection remains in the OWASP Top 10 and has been responsible for some of the largest data breaches in history.
- **Privilege escalation** – attackers exploit vulnerabilities or misconfigurations to gain higher-level database permissions than they are authorised to have. This can allow them to access restricted data, modify audit logs, or create new administrative accounts.
- **Denial of service** – overwhelming the database server with requests to make it unavailable to legitimate users. Resource-intensive queries, connection flooding, and exploitation of database vulnerabilities can all cause denial of service.
- **Data exfiltration** – the unauthorised transfer of data from the database to an external location. This can occur through SQL injection, compromised application accounts, insider threats, or malware with database access capabilities.
- **Unpatched vulnerabilities** – database vendors regularly release security patches addressing known vulnerabilities. Organisations that fail to apply patches in a timely manner leave their databases exposed to known exploits.
- **Default and weak credentials** – many database systems ship with default administrative accounts and passwords. Failure to change these defaults or enforce strong password policies provides trivial access for attackers.
- **Insider threats** – database administrators and other privileged users have extensive access to sensitive data. Malicious or negligent insiders can exfiltrate, modify, or destroy data without triggering external security controls.
- **Backup exposure** – database backups often contain complete copies of sensitive data but may not be protected with the same rigour as the production database. Unencrypted backups stored in insecure locations are a common source of data breaches.

Did you know?

According to the Verizon Data Breach Investigations Report, web application attacks – primarily SQL injection – are consistently among the top attack patterns leading to confirmed data breaches. The Heartland Payment Systems breach (2008), which compromised 130 million credit card records, was initiated through a SQL injection attack on the company's website. The attack went undetected for months, allowing the attackers to install packet-sniffing software on Heartland's network.

Case Study – The Equifax Data Breach (2017)

In September 2017, Equifax disclosed a data breach that exposed the personal information of 147 million consumers, including names, Social Security numbers, birth dates, addresses, and driver's licence numbers. The breach was caused by a known vulnerability in the Apache Struts web application framework (CVE-2017-5638) that had not been patched despite a fix being available for two months. Attackers exploited this vulnerability to access Equifax's databases, remaining undetected for 76 days.

Task: (1) What type of vulnerability enabled the initial access? (2) How does this breach illustrate the importance of patch management for database security? (3) What database-level controls could have limited the data exposed? (4) Equifax settled for approximately $700 million – calculate a per-record cost and compare it to industry averages. Write a 500-word analysis.

Over to you – Research Activity: Database Breaches

Research one of the following database breaches and prepare a 400-word briefing covering the attack vector, data compromised, impact, and lessons learned: (a) Capital One breach (2019), (b) Facebook/Meta data exposure (2021), (c) Microsoft Exchange Server vulnerabilities (2021), (d) Uber breach (2022).

Useful resource: https://haveibeenpwned.com/

1.2 Information security concepts and tools for database protection

Protecting databases requires a layered approach that combines technical, procedural, and physical controls. The effectiveness of these controls depends on how well they are implemented, maintained, and integrated into the organisation's overall security strategy.

Technical Controls

- **Encryption** – protects data confidentiality at rest (Transparent Data Encryption/TDE, column-level encryption) and in transit (TLS/SSL connections). Encryption ensures that even if data is stolen, it cannot be read without the decryption keys. Key management is critical – keys must be stored separately from the encrypted data.
- **Access controls** – implementing the principle of least privilege for database accounts. Use role-based access control (RBAC) to assign permissions based on job function. Separate application accounts from administrative accounts. Regularly review and audit permissions.

- **Database Activity Monitoring (DAM)** – real-time monitoring of all database activity, independent of native database audit functions. DAM solutions can detect SQL injection attempts, privilege escalation, data exfiltration, and policy violations. They provide alerts and detailed audit trails for forensic investigation.
- **Input validation and parameterised queries** – the primary defence against SQL injection. Parameterised queries (prepared statements) ensure that user input is treated as data, not executable code. Input validation filters and sanitises all user-supplied data before it reaches the database.
- **Database firewalls** – specialised firewalls that monitor SQL traffic and block queries that match known attack patterns or violate defined policies. Database firewalls provide an additional layer of protection between the application and the database.
- **Data masking and tokenisation** – replacing sensitive data with fictitious but realistic data (masking) for use in non-production environments, or replacing it with non-sensitive tokens (tokenisation) that can be mapped back to the original data through a secure token vault.
- **Audit logging** – comprehensive logging of all database activities including logins, queries, data modifications, schema changes, and administrative actions. Logs should be stored in a tamper-proof, centralised location and reviewed regularly.
- **Patch management** – systematically applying vendor security patches to database software. This requires a testing process to verify that patches do not break functionality, followed by timely deployment to production systems.

Procedural and Administrative Controls

- **Security policies and standards** – documented policies governing database security, including acceptable use, data classification, access management, change management, and incident response.
- **Separation of duties** – ensuring that no single individual has complete control over the database. For example, the DBA who administers the database should not be the same person who approves access requests or reviews audit logs.
- **Regular security assessments** – conducting periodic vulnerability assessments and penetration tests of database systems to identify and remediate weaknesses before they can be exploited.
- **Staff training and awareness** – educating developers, DBAs, and end users about database security risks and best practices. Developers, in particular, need training in secure coding practices to prevent SQL injection and other application-level vulnerabilities.
- **Data classification and handling** – categorising data based on sensitivity (e.g. public, internal, confidential, restricted) and applying appropriate security controls to each category. Not all data requires the same level of protection.

Industry Insight – Database Security in Financial Services

Financial institutions face some of the strictest database security requirements. The Payment Card Industry Data Security Standard (PCI DSS) mandates specific controls for databases storing cardholder data, including encryption, access controls, activity monitoring, and regular testing. Failure to comply can result in substantial fines and the loss of the ability to process card payments. PCI DSS v4.0, released in 2022, introduced

more flexible approaches to meeting security objectives while strengthening requirements for multi-factor authentication and encryption.

Read more: https://www.pcisecuritystandards.org/

Over to you – Database Security Audit Exercise

You are conducting a security audit of a small e-commerce company's MySQL database. Create a security audit checklist with at least 20 items covering: authentication and access controls, encryption, patch management, backup security, monitoring and logging, and application security. For each item, specify what you would check and what 'good' looks like.

Reading List

- Afyouni, H. (2020) *Database Security and Auditing: Protecting Data Integrity and Accessibility.* Boston, MA: Cengage Learning.
- Oracle (2024) *Oracle Database Security Guide.* Available at: https://docs.oracle.com/en/database/ (Accessed: 15 March 2026).
- Date, C.J. (2023) *An Introduction to Database Systems.* 9th edn. Harlow: Pearson.
- Shulman, A. (2022) *Database Activity Monitoring: Best Practices for Security and Compliance.* Sebastopol, CA: O'Reilly Media.
- Clarke, J. (2022) *SQL Injection Attacks and Defense.* 3rd edn. Waltham, MA: Syngress.

Summary

In this chapter, you have explored the security risks facing database systems, including SQL injection, privilege escalation, data exfiltration, insider threats, and backup exposure. You have assessed the effectiveness of technical controls (encryption, access controls, DAM, input validation, database firewalls, data masking, audit logging, and patch management) and procedural controls (security policies, separation of duties, security assessments, training, and data classification) in protecting databases. These foundational concepts underpin the more specific database management topics in the following chapters.

Chapter Two – Database Categories of Control

Introduction

This chapter examines the specific terminology used in database security and the categories of control that are applied to protect database systems. You will explore how databases are structured and managed, understand key concepts such as the Anderson Rule and the Grandfather-Father-Son backup model, and develop a working vocabulary of database security terminology.

Learning Outcomes

On completing the chapter, you will be able to:

1. **Understand types of database categories of control.**

Assessment Criteria

2.1 Explain database terminology and categories of control.

2.1 Database terminology and categories of control

Essential Database Terminology

Term	Definition and Security Relevance
Schema	The logical structure of the database, defining tables, columns, relationships, and constraints. Schema security prevents unauthorised structural modifications.
ACID Properties	Atomicity, Consistency, Isolation, Durability – the four properties that guarantee reliable database transactions. Essential for data integrity.
Stored Procedures	Pre-compiled SQL code stored in the database. When properly used, they can improve security by reducing SQL injection risk. When poorly secured, they can be exploited.
Triggers	Automated actions executed in response to specific database events (INSERT, UPDATE, DELETE). Used for audit logging and enforcing business rules.
Views	Virtual tables that present a subset of data from one or more tables. Views can be used to restrict access to sensitive columns or rows.
Normalisation	The process of organising data to reduce redundancy and improve integrity. Normalised databases have a smaller attack surface but may require more complex queries.
Replication	Copying data from one database server to another for redundancy and performance. Replication increases availability but also increases the number of locations where data must be secured.

Transaction Log	A sequential record of all database modifications. Critical for recovery, auditing, and forensic investigation after a security incident.

Categories of Database Control

Database security controls can be categorised into three broad categories that align with the defence-in-depth principle:

- **Technical controls** – implemented through technology and include authentication mechanisms (passwords, multi-factor authentication, certificates), authorisation systems (RBAC, row-level security), encryption (TDE, column-level, in-transit), audit mechanisms (native audit, DAM), and intrusion detection.
- **Administrative/procedural controls** – implemented through policies, processes, and human behaviour. These include security policies, access request and approval workflows, change management procedures, backup and recovery procedures, incident response plans, and regular security training.
- **Physical controls** – protect the physical infrastructure hosting database systems. These include data centre security (access controls, CCTV, environmental monitoring), locked server racks, secure disposal of storage media, and protection against environmental hazards (fire, flood, power loss).

! Need to know – The Anderson Rule

The Anderson Rule (named after computer security pioneer James P. Anderson) states that the security of a system depends on the security of its weakest component. In the context of database security, this means that protecting the database itself is insufficient if the application layer, network, operating system, or physical infrastructure is insecure. An attacker will target the weakest link in the chain. This principle reinforces the need for a holistic, defence-in-depth approach to database security.

The Grandfather-Father-Son (GFS) Backup Model

The Grandfather-Father-Son model is a widely used backup rotation scheme for disaster recovery. It organises backups into three tiers:

- **Son (daily)** – incremental or differential backups performed each working day. These capture changes since the last backup and are overwritten on a weekly cycle.
- **Father (weekly)** – full backups performed at the end of each week. These are retained for a monthly cycle and provide a known-good restore point for each week.
- **Grandfather (monthly)** – full backups taken at the end of each month and retained for a longer period (typically 12 months or more). These provide long-term recovery points for compliance and disaster recovery.

The GFS model provides a balance between storage efficiency, recovery speed, and data retention. For database security, it is essential that all backup tiers are encrypted, stored securely (including offsite/cloud copies), and regularly tested to verify successful restoration.

Example – GFS Backup Schedule

A company implements GFS for its critical customer database:

Daily (Son): Incremental backups at 23:00, Monday to Thursday. Friday's backup becomes the weekly (Father). Monthly (Grandfather): First Friday of each month's full backup is retained for 12 months. All backups encrypted with AES-256 and replicated to a secondary data centre.

If the database is corrupted on Wednesday afternoon, recovery involves: (1) restore last Friday's full backup, (2) apply Monday's incremental, (3) apply Tuesday's incremental. Data loss is limited to changes made on Wednesday before the corruption.

Over to you – Video Watch: Database Backup Strategies

Title: Backup and Recovery Strategies Explained – CBT Nuggets

Link: https://www.youtube.com/watch?v=01SXrlWMUWA

After watching, design a backup strategy for a hospital patient records database. Specify: backup types, frequency, retention periods, encryption, storage locations, and testing schedule. Explain how your strategy balances RTO and RPO requirements.

Over to you – Database Design Activity

Using a free tool such as dbdiagram.io (https://dbdiagram.io/), design a simple relational database schema for an online bookshop. Include tables for: customers, orders, order items, books, and authors. Define primary keys, foreign keys, and data types. Then identify three potential security risks in your design and propose mitigations for each.

Reading List

- Connolly, T. and Begg, C. (2023) *Database Systems: A Practical Approach to Design, Implementation, and Management.* 7th edn. Harlow: Pearson.
- Elmasri, R. and Navathe, S.B. (2022) *Fundamentals of Database Systems.* 8th edn. Harlow: Pearson.
- Silberschatz, A., Korth, H.F. and Sudarshan, S. (2024) *Database System Concepts.* 8th edn. New York: McGraw-Hill.
- Murray, W.H. and Cegielski, C.G. (2021) *Database Management: An Organizational Perspective.* New York: Prospect Press.

Summary

In this chapter, you have explored essential database terminology and the three categories of database control: technical, administrative, and physical. You have learned about the Anderson Rule and its implications for holistic security, and examined the Grandfather-Father-Son backup model as a disaster recovery strategy. These concepts provide the framework for understanding how databases are protected in practice.

Chapter Three – Cloud-Based Storage and Database Tools

Introduction

This chapter explores the concepts and models of cloud-based storage solutions and investigates the database tools available to different stakeholders including data owners, data custodians, incident responders, and investigators. Cloud computing has fundamentally transformed how organisations store and manage data, introducing both new capabilities and new security challenges.

Learning Outcomes

On completing the chapter, you will be able to:

1. **Understand the underpinning concepts and models of cloud-based storage solutions.**

Assessment Criteria

3.1 Explore the functionality of database tools available to Data Owners, Custodians, Incident Responders and Investigators.

3.1 Database tools for data owners, custodians and investigators

Cloud Computing Models

Cloud computing delivers computing resources as a service over the internet. The three primary service models are:

- **Infrastructure as a Service (IaaS)** – provides virtualised computing resources (servers, storage, networking). The customer manages the operating system, applications, and data. Examples: Amazon EC2, Microsoft Azure VMs, Google Compute Engine.
- **Platform as a Service (PaaS)** – provides a platform for developing and deploying applications without managing the underlying infrastructure. Examples: AWS Elastic Beanstalk, Azure App Service, Google App Engine.
- **Software as a Service (SaaS)** – provides complete applications over the internet. The provider manages everything. Examples: Salesforce, Microsoft 365, Google Workspace.

Additionally, Database as a Service (DBaaS) is a growing model where the cloud provider manages the database infrastructure, patching, backups, and high availability. Examples include Amazon RDS, Azure SQL Database, and Google Cloud SQL.

> **! Need to know – The Shared Responsibility Model**
>
> In cloud computing, security is a shared responsibility between the cloud provider and the customer. The provider is responsible for the security OF the cloud (physical infrastructure, hypervisor, network). The customer is responsible for security IN the cloud

(data, access management, application configuration, encryption keys). Misunderstanding this model is one of the leading causes of cloud security incidents. In a DBaaS model, the provider manages database patching and infrastructure, but the customer is still responsible for access controls, encryption, data classification, and compliance.

Cloud Database Security Challenges

- **Data sovereignty and compliance** – data stored in the cloud may reside in different geographic jurisdictions, raising legal questions about which data protection laws apply. UK GDPR requires that personal data of UK residents is adequately protected regardless of where it is stored.
- **Multi-tenancy risks** – cloud databases share physical infrastructure with other customers. Logical isolation prevents cross-tenant data access, but vulnerabilities in the virtualisation layer could theoretically allow breaches.
- **Visibility and control** – organisations have less visibility into the underlying infrastructure of cloud databases, making it harder to verify security configurations and detect threats.
- **Identity and access management** – managing access to cloud databases across distributed teams requires robust IAM policies, federated authentication, and consistent enforcement of least privilege.
- **Data migration and lock-in** – moving sensitive data to and from cloud databases creates exposure during migration. Proprietary cloud features can create vendor lock-in, making it difficult to change providers.

Case Study – Capital One Cloud Data Breach (2019)

In July 2019, a former Amazon Web Services employee exploited a misconfigured web application firewall to access Capital One's cloud-hosted databases on AWS. The breach exposed personal information of over 100 million customers and credit card applicants, including names, addresses, credit scores, and Social Security numbers. The root cause was a Server-Side Request Forgery (SSRF) vulnerability combined with overly permissive IAM roles.

Task: (1) How does this breach illustrate the shared responsibility model? (2) What IAM misconfigurations contributed to the breach? (3) How could network segmentation and least privilege have limited the damage? (4) What are the implications for organisations considering cloud database migration? Write a 500-word analysis.

Database Tools for Different Stakeholders

Different roles within an organisation require different database tools and capabilities:

- **Data Owners** – business stakeholders responsible for data classification and access decisions. Tools: data catalogues (Apache Atlas, Collibra), data governance platforms, classification frameworks, and consent management tools.
- **Data Custodians/DBAs** – technical staff responsible for the day-to-day management and security of database systems. Tools: database management consoles (phpMyAdmin, SQL Server Management Studio, pgAdmin), backup and recovery

tools, performance monitoring (SolarWinds DPA, Datadog), and configuration management tools.

- **Incident Responders** – security professionals who investigate and respond to database security incidents. Tools: database activity monitoring (Imperva, IBM Guardium), log analysis platforms (Splunk, ELK Stack), forensic imaging tools, and query analysis tools.
- **Investigators/Forensics** – specialists who conduct detailed analysis of database breaches after they occur. Tools: forensic database analysis tools, transaction log analysers, data recovery tools, timeline reconstruction software, and evidence collection platforms that maintain chain of custody.

Industry Insight – Big Data and Database Security

The volume of data generated globally is growing exponentially. By 2025, the total amount of data created, captured, copied, and consumed globally was projected to reach 181 zettabytes. Big data platforms (Hadoop, Spark, data lakes) introduce unique security challenges including scale-appropriate access controls, data lineage tracking, and securing distributed processing across clusters. Traditional database security tools often struggle with the volume, velocity, and variety of big data, requiring purpose-built solutions.

Over to you – Cloud Security Comparison Activity

Compare the database security features of two major cloud providers (AWS RDS vs Azure SQL Database OR Google Cloud SQL). For each, investigate: encryption options, access control mechanisms, audit logging capabilities, backup and recovery features, compliance certifications, and pricing model. Present your findings in a comparison table with a 300-word recommendation.

Over to you – Video Watch: Cloud Security

Title: Cloud Security – What is the Shared Responsibility Model? – IBM Technology

Link: https://youtu.be/jl8lKpjiCSM?si=5rNzMh7SFslPpARJ

After watching, create a diagram showing the shared responsibility model for a DBaaS deployment. Clearly label which security responsibilities belong to the provider and which belong to the customer.

Reading List

- Mather, T., Kumaraswamy, S. and Latif, S. (2023) *Cloud Security and Privacy: An Enterprise Perspective on Risks and Compliance.* 2nd edn. Sebastopol, CA: O'Reilly Media.
- CSA (2024) *Cloud Controls Matrix v4.* Cloud Security Alliance. Available at: https://cloudsecurityalliance.org/research/cloud-controls-matrix/ (Accessed: 15 March 2026).
- Raj, P. and Raman, A. (2022) *Handbook of Research on Cloud Computing and Big Data Applications in IoT.* Hershey, PA: IGI Global.

Summary

In this chapter, you have explored cloud computing models and their implications for database security. You have examined the shared responsibility model, cloud-specific security challenges, and the tools available to data owners, custodians, incident responders, and investigators. Understanding these concepts is essential as organisations increasingly migrate their databases to cloud environments.

Chapter Four – Computer Programming and Hacking

Introduction

This chapter explores the fundamental relationship between computer programming and computer hacking. Understanding how software is created is essential for understanding how it can be exploited. You will examine popular programming languages, understand the distinction between compiled and interpreted languages, and analyse the symbiotic relationship between advances in programming and the evolution of hacking techniques.

Learning Outcomes

On completing the chapter, you will be able to:

1. **Understand the relationship between computer programming and computer hacking.**

Assessment Criteria

4.1 Explain various popular computer programming languages.

4.2 Analyse the relationship between programming skills and the ability to hack into systems.

4.1 Popular computer programming languages

Compiled vs Interpreted Languages

A fundamental distinction in programming languages is between compiled and interpreted languages:

- **Compiled languages** – source code is translated entirely into machine code (binary) by a compiler before execution. The resulting executable runs directly on the hardware. Examples: C, C++, Rust, Go. Compiled code is generally faster and harder to reverse-engineer, but the compilation step slows development and executables are platform-specific.
- **Interpreted languages** – source code is executed line by line by an interpreter at runtime. No separate compilation step is needed. Examples: Python, JavaScript, Ruby, PHP. Interpreted code is more portable and easier to develop with but generally slower and easier to analyse (since the source code is often available).
- **Hybrid approaches** – some languages use a combination. Java compiles to bytecode that runs on the Java Virtual Machine (JVM). C# compiles to Common Intermediate Language (CIL) for the .NET runtime. These approaches aim to balance performance with portability.

Programming Languages and Their Security Relevance

Language	Characteristics	Security Relevance
C/C++	Low-level, high performance, direct memory access. The foundation of	Vulnerable to buffer overflows, memory corruption, and use-after-free bugs. Many critical

	operating systems, firmware, and embedded systems.	vulnerabilities (Heartbleed, EternalBlue) originated in C/C++ code.
Python	High-level, interpreted, versatile. Extensive library ecosystem. Rapid prototyping.	The most popular language for security tools, exploit development, automation, and penetration testing. Used by both defenders and attackers.
JavaScript	The language of the web. Runs in browsers and on servers (Node.js).	Central to web application security. XSS attacks exploit JavaScript. Also used for security testing tools and browser exploitation.
Java	Platform-independent (JVM), enterprise-focused. Widely used in large organisations.	Enterprise attack surface. The Log4Shell vulnerability (2021) in the Java logging library Log4j affected millions of systems globally.
SQL	Domain-specific language for database operations.	SQL injection is among the most common and damaging attack vectors. Understanding SQL is essential for both attacking and defending databases.
PowerShell	Windows system administration and automation language.	Heavily used in post-exploitation and living-off-the-land attacks. PowerShell commands can download and execute payloads, exfiltrate data, and modify system configurations.
Bash	Unix/Linux command-line scripting language.	Used for system administration and automation on Linux servers. Shellshock (2014) was a critical Bash vulnerability that affected millions of systems.
Rust	Systems programming language with memory safety guarantees.	Increasingly adopted for security-critical software. Prevents entire classes of memory-related vulnerabilities that plague C/C++.
Go	Developed by Google. Compiled, concurrent, efficient.	Popular for developing modern security tools, malware, and cloud-native applications. Used in tools like Docker and Kubernetes.
Assembly	Low-level language with direct hardware access.	Essential for reverse engineering, malware analysis, and exploit development. Understanding assembly is key to analysing compiled malware.

Over to you – Video Watch: Programming Languages

Title: 100+ Computer Science Concepts Explained – Fireship

Link: https://www.youtube.com/watch?v=-uleG_Vecis

After watching, select three programming languages and research one major security vulnerability or attack associated with each. Present your findings in a comparison table.

4.2 The relationship between programming skills and hacking

The relationship between programming and hacking is symbiotic: every advance in programming creates new capabilities that can be used for both constructive and destructive purposes. Understanding this relationship is essential for cyber security professionals.

Why Hackers Need Programming Skills

- **Exploit development** – creating custom exploits for newly discovered vulnerabilities requires deep understanding of the target language, memory management, and system architecture.
- **Tool customisation** – while many hacking tools are available off-the-shelf, sophisticated attackers customise and modify tools to evade detection and target specific environments.
- **Automation** – programming skills enable attackers to automate reconnaissance, scanning, exploitation, and data exfiltration, dramatically increasing the scale and speed of attacks.
- **Reverse engineering** – understanding how software works at the code level allows attackers to discover vulnerabilities, bypass protections, and create targeted exploits.
- **Evasion** – programming skills enable attackers to modify malware to evade antivirus signatures, obfuscate network traffic, and bypass security controls.

Why Defenders Need Programming Skills

- **Security tool development** – creating custom scripts for log analysis, threat detection, vulnerability scanning, and incident response.
- **Code review** – identifying vulnerabilities in application source code before they can be exploited.
- **Automation of defences** – scripting firewall rules, automating patch deployment, and building security orchestration workflows.
- **Malware analysis** – understanding malicious code to determine its capabilities, extract indicators of compromise (IOCs), and develop countermeasures.
- **Penetration testing** – writing custom exploits and testing tools to evaluate an organisation's security posture.

Did you know?

The term 'hacker' originally had a positive meaning. In the 1960s at MIT, a 'hack' was an elegant or creative solution to a technical problem, and 'hackers' were highly skilled programmers who pushed the boundaries of what computers could do. The term only acquired its negative connotation in the 1980s and 1990s as computer crime became

more prevalent. Today, the security community distinguishes between 'white hat' (ethical) hackers, 'black hat' (malicious) hackers, and 'grey hat' hackers who operate in a moral grey area.

Over to you – Ethical Hacking Research Task

Research the concept of 'bug bounty' programmes. Answer: (a) What is a bug bounty programme? (b) Name three major companies that run bug bounty programmes. (c) What programming skills are most commonly needed for bug bounty hunting? (d) What is the difference between responsible disclosure and full disclosure? Write a 400-word briefing note.

Reading List

- Erickson, J. (2024) *Hacking: The Art of Exploitation.* 3rd edn. San Francisco, CA: No Starch Press.
- Kennedy, D., O'Gorman, J., Kearns, D. and Aharoni, M. (2022) *Metasploit: The Penetration Tester's Guide.* 2nd edn. San Francisco, CA: No Starch Press.
- Seitz, J. and Arnold, T. (2021) *Black Hat Python: Python Programming for Hackers and Pentesters.* 2nd edn. San Francisco, CA: No Starch Press.
- OccupyTheWeb (2023) *Linux Basics for Hackers: Getting Started with Networking, Scripting, and Security in Kali Linux.* 2nd edn. San Francisco, CA: No Starch Press.

Summary

In this chapter, you have explored the distinction between compiled and interpreted programming languages, examined the security relevance of ten popular programming languages, and analysed the symbiotic relationship between programming skills and hacking. You have seen that both attackers and defenders require programming skills, and that understanding code is fundamental to identifying, exploiting, and mitigating software vulnerabilities.

Chapter Five – Python: The Hacker's Language

Introduction

This chapter provides a focused examination of Python as a programming language of particular significance to the cyber security profession. Python has become the language of choice for both ethical hackers and malicious actors due to its readability, extensive library ecosystem, rapid development capabilities, and cross-platform compatibility. You will investigate how Python is used in both non-malicious and malicious hacking contexts.

Learning Outcomes

On completing the chapter, you will be able to:

1. **Understand the 'interpreted' general-purpose programming language, Python.**

Assessment Criteria

5.1 Investigate where non-malicious and malicious hackers have utilised Python.

5.1 Python in non-malicious and malicious hacking

Why Python Dominates Cyber Security

Python has become the dominant programming language in cyber security for several compelling reasons:

- **Readability and simplicity** – Python's clean syntax makes it accessible to security professionals who may not be full-time developers. Code can be written and understood quickly, which is essential during time-sensitive security incidents.
- **Extensive library ecosystem** – Python has thousands of libraries specifically designed for security tasks, including Scapy (network packet manipulation), Requests (HTTP operations), Beautiful Soup (web scraping), Cryptography (encryption), and Paramiko (SSH connections).
- **Rapid prototyping** – Python enables security professionals to quickly build proof-of-concept tools, automate repetitive tasks, and test hypotheses without the overhead of compilation.
- **Cross-platform compatibility** – Python runs on Windows, Linux, and macOS, making tools written in Python portable across the operating systems commonly encountered in security work.
- **Community and resources** – an enormous community of security-focused Python developers contributes tools, libraries, tutorials, and documentation, creating a rich ecosystem for learning and development.

Python in Ethical (Non-Malicious) Hacking

Ethical hackers and security professionals use Python extensively across all phases of security work:

- **Penetration testing frameworks** – major frameworks like Metasploit have Python interfaces. Tools like SQLMap (automated SQL injection), Impacket (network protocol manipulation), and Volatility (memory forensics) are written in Python.

- **Network scanning and reconnaissance** – Python scripts using libraries like Scapy, Nmap (via python-nmap), and Socket can automate network discovery, port scanning, and service identification.
- **Web application testing** – tools built with Requests, Beautiful Soup, and Selenium automate the testing of web applications for vulnerabilities such as XSS, CSRF, and broken authentication.
- **Log analysis and threat hunting** – Python's data processing capabilities (Pandas, NumPy) make it ideal for analysing large volumes of security logs to identify patterns and anomalies.
- **Automation and orchestration** – security operations teams use Python to automate incident response workflows, integrate security tools, and build custom dashboards and reporting systems.
- **Malware analysis** – Python tools like YARA (malware classification), pefile (PE file analysis), and Capstone (disassembly) are used to analyse and classify malicious software.

Example – Simple Python Port Scanner

The following pseudocode illustrates how a basic port scanner works in Python:

```
import socket
target = "192.168.1.1"
for port in range(1, 1025):
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    result = sock.connect_ex((target, port))
    if result == 0: print(f"Port {port} is open")
    sock.close()
```

This simple script attempts to connect to each port on the target. A return value of 0 indicates the port is open. In practice, security professionals use more sophisticated tools, but understanding the underlying logic is important.

Note: Only scan systems you own or have written permission to test.

Python in Malicious Hacking

Unfortunately, the same qualities that make Python attractive to security professionals also make it attractive to malicious actors:

- **Rapid exploit development** – attackers use Python to quickly develop proof-of-concept exploits for newly disclosed vulnerabilities, sometimes within hours of a CVE being published.
- **Custom malware** – Python can be used to create custom malware including keyloggers, backdoors, and ransomware. Tools like PyInstaller can package Python scripts as standalone executables, making distribution easier.
- **Phishing tools** – Python scripts can automate the creation and sending of phishing emails, clone legitimate websites, and harvest stolen credentials.
- **Botnet command and control** – Python's networking capabilities make it suitable for building command-and-control infrastructure for botnets.
- **Data exfiltration** – Python scripts can automate the extraction of data from compromised systems, encoding it to evade detection and transmitting it to attacker-controlled servers.

Case Study – Python-Based Malware in the Wild

In 2022, security researchers identified a campaign dubbed 'PyPI Malware' where attackers uploaded malicious Python packages to the Python Package Index (PyPI), the official repository for Python libraries. These packages had names similar to popular libraries (a technique called 'typosquatting') and contained hidden malware that stole credentials, cryptocurrency wallet data, and system information when installed. Over 10,000 downloads occurred before the packages were detected and removed.

Task: (1) What is 'typosquatting' and why is it effective? (2) How could a developer protect themselves from installing malicious packages? (3) What verification mechanisms does PyPI provide, and are they sufficient? (4) Research one other instance of supply chain attacks via package repositories (e.g. npm, RubyGems). Write a 500-word comparative analysis.

Over to you – Video Watch: Python for Cyber Security

Title: Python for Cybersecurity – Full Course for Beginners – freeCodeCamp

Link:
https://www.youtube.com/watch?v=eWRfhZUzrAc&list=PLWKjhJtqVAbnqBxcdjVGgT3uVR10bzTEB

Watch the first 30 minutes and then write a 300-word reflection on how Python's features make it particularly well-suited for cyber security tasks. Include at least two specific examples of Python libraries and their security applications.

Over to you – Hands-On Python Activity

Install Python on your computer (https://www.python.org/downloads/). Complete the following exercises:

(a) Write a Python script that generates a random 16-character password containing uppercase letters, lowercase letters, numbers, and special characters. (b) Write a Python script that takes a file path as input and calculates its SHA-256 hash (use the hashlib library). (c) Write a Python script that reads a CSV file of IP addresses and checks whether each is a valid IPv4 address. (d) Research the 'Requests' library and use it to check whether a list of websites return HTTP 200 status codes.

Document your code with comments explaining what each section does.

Over to you – Research: Python Security Tools

Research and write a 500-word report on three of the following Python-based security tools. For each tool, explain: its purpose, key features, how it is used in practice, and whether it can be used for both offensive and defensive purposes.

Tools: SQLMap, Scapy, Volatility, Impacket, Nmap (python-nmap), YARA, Burp Suite extensions, Responder, BloodHound.

Reading List

- Seitz, J. and Arnold, T. (2021) *Black Hat Python: Python Programming for Hackers and Pentesters.* 2nd edn. San Francisco, CA: No Starch Press.
- Sweigart, A. (2023) *Automate the Boring Stuff with Python.* 3rd edn. San Francisco, CA: No Starch Press.
- Matthes, E. (2023) *Python Crash Course: A Hands-On, Project-Based Introduction to Programming.* 3rd edn. San Francisco, CA: No Starch Press.
- OccupyTheWeb (2023) *Linux Basics for Hackers.* 2nd edn. San Francisco, CA: No Starch Press.
- Lutz, M. (2022) *Learning Python.* 6th edn. Sebastopol, CA: O'Reilly Media.

Summary

In this chapter, you have investigated Python's dominance in the cyber security landscape, examining why its characteristics make it the language of choice for both ethical and malicious hackers. You have explored how Python is used in penetration testing, network analysis, web application testing, log analysis, and malware analysis. You have also examined how the same capabilities are exploited by malicious actors for exploit development, custom malware, phishing, and data exfiltration. Understanding Python's role in cyber security is essential for any professional working in this field.

Glossary

Word / Term	Explanation
ACID	Atomicity, Consistency, Isolation, Durability – properties guaranteeing reliable database transactions.
Anderson Rule	The principle that a system's security depends on the security of its weakest component.
Buffer Overflow	A vulnerability where a program writes data beyond the allocated memory boundary.
Cloud (IaaS/PaaS/SaaS)	Service models for delivering computing resources over the internet.
Compiled Language	A programming language translated entirely into machine code before execution.
DAM	Database Activity Monitoring – real-time monitoring of all database operations.
Data Exfiltration	Unauthorised transfer of data from a database or system to an external location.
DBaaS	Database as a Service – cloud-managed database infrastructure.
GFS Backup	Grandfather-Father-Son – a tiered backup rotation scheme for disaster recovery.
Interpreted Language	A programming language executed line by line by an interpreter at runtime.
NoSQL	Non-relational database systems designed for flexible schemas and specific data models.
Parameterised Query	A pre-compiled SQL statement where user input is treated as data, preventing injection.
PCI DSS	Payment Card Industry Data Security Standard – security requirements for cardholder data.
Privilege Escalation	Gaining higher-level permissions than authorised, often through exploitation.
PyPI	Python Package Index – the official repository for Python packages and libraries.
Python	A high-level, interpreted programming language widely used in cyber security.
RDBMS	Relational Database Management System – a database system based on the relational model.
Shared Responsibility	Cloud security model where provider and customer share security obligations.
SQL	Structured Query Language – the standard language for managing relational databases.

SQL Injection	An attack inserting malicious SQL through application inputs to manipulate databases.
Stored Procedure	Pre-compiled SQL code stored in the database for reuse and security.
TDE	Transparent Data Encryption – encrypts database files at the storage level.
Tokenisation	Replacing sensitive data with non-sensitive tokens mapped through a secure vault.
Typosquatting	Registering names similar to popular packages/domains to trick users into downloading malware.
View	A virtual table presenting a subset of data, used to restrict access to sensitive information.

MCQs and True & False Questions (self-assessment)

True or False Questions

1. SQL injection is an attack that targets database systems through application inputs.
2. NoSQL databases are immune to security vulnerabilities.
3. The Anderson Rule states that security depends on the weakest component.
4. The Grandfather-Father-Son model is a network architecture design.
5. Transparent Data Encryption (TDE) encrypts data in transit only.
6. Parameterised queries are the primary defence against SQL injection.
7. In the shared responsibility model, the cloud provider is responsible for all security.
8. Python is a compiled programming language.
9. Database Activity Monitoring operates independently of native database audit functions.
10. ACID properties ensure reliable database transactions.
11. Data masking replaces sensitive data with realistic fictitious data.
12. C/C++ programs are vulnerable to buffer overflow attacks.
13. The term ‘hacker’ has always had a negative connotation.
14. SQLMap is a Python-based tool for automated SQL injection testing.
15. Typosquatting involves creating package names similar to popular libraries.
16. Separation of duties means one person should manage all database functions.
17. Views can be used to restrict access to sensitive database columns.
18. PCI DSS applies to organisations that store cardholder data.
19. An interpreted language requires compilation before execution.
20. Python’s Scapy library is used for network packet manipulation.
21. Database backups should be encrypted and stored securely.
22. PowerShell is commonly used in living-off-the-land attacks on Windows systems.
23. The Capital One breach was caused by a misconfigured web application firewall.
24. Data sovereignty refers to the legal jurisdiction governing stored data.
25. Rust was designed to prevent memory-related vulnerabilities.

Multiple Choice Questions

1. Which is the most common database attack vector?

A. DDoS
B. SQL injection
C. Phishing
D. Brute force

2. What does TDE stand for?

A. Total Data Encryption
B. Transparent Data Encryption
C. Transfer Data Encoding
D. Temporal Data Extraction

3. The GFS backup model includes which three tiers?

A. Gold, Silver, Bronze
B. Grandfather, Father, Son
C. Global, Federal, State
D. General, Focused, Specific

4. Python is classified as:

A. A compiled language
B. A markup language
C. An interpreted language
D. A query language

5. Which property is NOT part of ACID?

A. Atomicity
B. Availability
C. Consistency
D. Durability

6. The shared responsibility model applies to:

A. On-premises databases only
B. Cloud computing environments
C. Physical security only
D. Open-source software

7. Which Python library is used for network packet manipulation?

A. Pandas
B. NumPy
C. Scapy
D. Flask

8. What vulnerability caused the Equifax breach?

A. SQL injection
B. Unpatched Apache Struts
C. Weak passwords
D. Physical intrusion

9. Database firewalls monitor:

A. Physical access to servers
B. SQL traffic between applications and databases
C. Internet bandwidth
D. Email communications

10. Which language is most associated with buffer overflow vulnerabilities?

A. Python
B. JavaScript
C. C/C++
D. Ruby

11. Tokenisation differs from encryption because:

A. It is reversible
B. Tokens cannot be mathematically derived from original data
C. It uses symmetric keys
D. It only works with numbers

12. Log4Shell (2021) was a vulnerability in:

A. Python
B. PHP
C. Java (Log4j)
D. Ruby on Rails

13. Data Owners are primarily responsible for:

A. Installing database patches
B. Data classification and access decisions
C. Physical server maintenance
D. Writing SQL queries

14. Which tool is used for memory forensics?

A. Wireshark
B. Nmap
C. Volatility
D. Metasploit

15. The primary defence against SQL injection is:

A. Firewalls
B. Encryption
C. Parameterised queries
D. Antivirus software

16. Which cloud model requires the customer to manage the most?

A. SaaS
B. PaaS
C. IaaS
D. DBaaS

17. PyPI is:

A. A Python IDE
B. The official Python package repository
C. A database tool
D. A cloud platform

18. Separation of duties in database security means:

A. Using multiple databases
B. Dividing critical tasks among multiple people
C. Separating data into different tables
D. Using different programming languages

19. Which is NOT a type of NoSQL database?

A. Document

B. Key-value
C. Relational
D. Graph

20. The Anderson Rule reinforces which security principle?

A. Single sign-on
B. Defence in depth
C. Open-source development
D. Cloud-first strategy

Answers to True/False Questions

1. *True.* SQL injection manipulates databases through unsanitised application inputs.

2. *False.* NoSQL databases have their own vulnerability categories, including injection attacks specific to their query languages.

3. *True.* The Anderson Rule states that a system's security is only as strong as its weakest component.

4. *False.* GFS is a backup rotation scheme for disaster recovery, not a network architecture.

5. *False.* TDE encrypts data at rest (stored on disk), not in transit.

6. *True.* Parameterised queries ensure user input is treated as data, not executable SQL.

7. *False.* Security is shared: the provider secures the cloud infrastructure; the customer secures their data and access.

8. *False.* Python is an interpreted language; source code is executed line by line at runtime.

9. *True.* DAM operates independently, providing an additional layer of monitoring beyond native database auditing.

10. *True.* ACID (Atomicity, Consistency, Isolation, Durability) guarantees transaction reliability.

11. *True.* Data masking creates realistic but fictitious data for non-production environments.

12. *True.* C/C++ allows direct memory access, making programs vulnerable to buffer overflow attacks.

13. *False.* The term originally meant a skilled, creative programmer. It acquired negative connotations in the 1980s–1990s.

14. *True.* SQLMap is a widely used open-source Python tool for detecting and exploiting SQL injection.

15. *True.* Typosquatting relies on developers mistyping package names during installation.

16. *False.* Separation of duties divides critical functions among multiple people to prevent abuse.

17. *True.* Database views present filtered subsets of data, restricting access to sensitive fields.

18. *True.* PCI DSS mandates specific security controls for organisations handling payment card data.

19. *False.* Interpreted languages are executed directly by an interpreter without a separate compilation step.

20. *True.* Scapy enables creation, manipulation, and analysis of network packets in Python.

21. *True.* Unencrypted backups in insecure locations are a common source of data breaches.

22. *True.* PowerShell is frequently abused for post-exploitation activities on Windows systems.

23. *True.* A misconfigured WAF combined with overly permissive IAM roles enabled the Capital One breach.

24. *True.* Data sovereignty concerns which legal jurisdiction's laws apply to data stored in different countries.

25. *True.* Rust's ownership and borrowing system prevents memory-related vulnerabilities at compile time.

Answers to Multiple Choice Questions

1. (B) SQL injection
2. (B) Transparent Data Encryption
3. (B) Grandfather, Father, Son
4. (C) An interpreted language
5. (B) Availability
6. (B) Cloud computing environments
7. (C) Scapy
8. (B) Unpatched Apache Struts
9. (B) SQL traffic between applications and databases
10. (C) C/C++
11. (B) Tokens cannot be mathematically derived from original data
12. (C) Java (Log4j)
13. (B) Data classification and access decisions
14. (C) Volatility
15. (C) Parameterised queries
16. (C) IaaS
17. (B) The official Python package repository
18. (B) Dividing critical tasks among multiple people
19. (C) Relational
20. (B) Defence in depth