MODULE 9 · STUDY NOTES

Securing User, Application, and Data Access

IAM groups and ABAC, identity federation with AWS STS and Amazon Cognito, multi-account governance with AWS Organizations and SCPs, encrypting data at rest with AWS KMS, and the AWS security service portfolio

Every AWS architecture eventually has to answer three questions: who is this user, what may they touch, and how is the data protected while it sits still. This module works through all three. It begins inside a single account, where **IAM groups** and **attribute-based access control (ABAC)** replace the unscalable habit of attaching a policy to every individual user. It then reaches outside the account with **identity federation** — letting an external identity provider vouch for a user so **AWS STS** can hand back temporary credentials — and with **Amazon Cognito**, which adds sign-up, sign-in, and AWS access to web and mobile applications. It zooms out to the whole organization, where **AWS Organizations** consolidates many accounts and **service control policies (SCPs)** and **permissions boundaries** set guardrails no local administrator can exceed. Finally it turns to the data itself — symmetric, asymmetric, and envelope encryption, client- versus server-side encryption, and **AWS KMS** as the managed home for the keys — before closing with the wider catalog of AWS security services that put defense in depth into practice.

LEARNING OBJECTIVES

- Use AWS Identity and Access Management (IAM) users, groups, and roles to manage permissions
- Implement user federation within an architecture to increase security
- Describe how to manage multiple AWS accounts
- Recognize how AWS Organizations service control policies (SCPs) increase security within an architecture
- Encrypt data at rest by using AWS Key Management Service (AWS KMS)
- Identify appropriate AWS security services based on a given use case

9.1 Managing permissions

Attaching a policy directly to every IAM user does not scale. Imagine three developers — John, Mary, and Pat — who each have an individual policy granting access to Amazon EC2. To also give them Amazon S3 access, an administrator must edit **three separate policies**, one per user. With 100 developers, that is 100 edits for a single change. The fix is the **IAM group**: a collection of IAM users to which you attach one policy so that every member inherits the permissions. Build groups around **job functions** — admins, developers, testers — and a new hire simply joins the right group and immediately gains the same access as their peers; someone changing roles is moved from one group to another.

CHARACTERISTICS OF AN IAM GROUP

- You can add users to or remove them from a group at any time.
- A user can belong to **multiple groups**, but groups **cannot be nested** — a group cannot contain another group.
- Groups are granted permissions through access-control policies attached to the group.
- Groups have **no security credentials** and cannot access AWS services directly — they exist only to make managing user permissions easier.
- Permissions in a policy attached directly to a user override group permissions when they are more restrictive.

WORKED EXAMPLE – A USER POLICY AND A GROUP POLICY TOGETHER

Zhang has a **user policy** that allows Amazon S3 and Amazon DynamoDB but explicitly **denies** Amazon Kinesis. Zhang is also in a **Developer group** whose policy allows S3, Amazon Athena, DynamoDB, and Kinesis. Can Zhang use Athena? **Yes** — the group allows it and nothing denies it, even though the user policy never mentions Athena. Can Zhang use Kinesis? **No** — the explicit deny in the user policy overrides the group's allow. The rule to carry into every exam question: **an explicit deny always overrides an explicit allow.**

Groups organize *who* gets a policy, but they do not change *how* the policy is written. The traditional model is **role-based access control (RBAC)**: you define permissions by job function in a policy that **lists the individual resources** the role may use, then attach it to an IAM entity (user, group, or role). Its weakness shows when the environment grows — when a new resource appears, you must **update every policy** that should reach it, and a resource used by several roles means editing several policies. That maintenance burden is what **attribute-based access control (ABAC)** removes.

ABAC defines permissions based on attributes. An attribute is a key or a key-value pair, and in AWS those attributes are **tags**, which can be applied both to IAM resources (users and roles) and to AWS resources. An ABAC policy checks whether an attribute on the identity also appears on the resource being accessed — so you can grant developers access to their project's resources **without naming any resource in the policy**. When you create a new user or a new resource, you simply apply the correct tags. The benefits: it is more flexible than listing each resource, it supports **granular permissions without a policy update for every new user or resource**, it is **highly scalable**, and it is **fully auditable**.

APPLYING ABAC IN FOUR STEPS

- **Set access-control attributes on identities** — tag IAM users or roles (for example, `Env = Dev`, `Project = Maint`).
- **Require attributes on new resources** — write policies that enforce required tags at resource-creation time.
- **Configure permissions based on attributes** — allow identities whose tags match the resource's tags, and deny those without the required tags.
- **Use one policy for all permissions** — adding a resource means just tagging it; matching roles gain access with no policy change.

KEY TERMS

Tagging limits
Up to 50 tags per resource; a key can be up to 128 Unicode characters and a value up to 256; keys and values are case-sensitive

Reserved prefix
Do not use `aws:` as a tag-key prefix — it is reserved for AWS use

Practical uses of tags
Cost allocation and billing, filtered resource views, and access control (ABAC)

TABLE 9.1 RBAC vs. ABAC at a glance

DIMENSION	RBAC	ABAC
Basis for access	Job-function role; policy lists individual resources	Attributes (tags) matched between identity and resource
Policies to maintain	Multiple — one per role, updated per resource change	One policy that scales across users and resources
Adding a new resource	Update every affected policy	Just apply the right tag to the resource
Scalability	Limited; maintenance grows with the environment	High, granular, and fully auditable

COMMON PITFALL

Do not confuse *organizing* permissions with *scaling* them. Groups solve the problem of applying one policy to many users, but a group still uses an RBAC-style policy that lists resources. When a scenario stresses rapid growth in users **and** resources, the scalable answer is **ABAC**, which combines permissions into a single attribute-driven policy.

9.2 Federating users

Identity federation is a system of trust between two parties to authenticate users and convey the information needed to authorize access to resources. The **identity provider (IdP)** is responsible for authenticating the user; the **service provider (SP)** is responsible for controlling access to its resources. Through administrative agreement and configuration, the SP trusts the IdP's authentication and grants access accordingly. IdPs come in two common flavors: **OpenID Connect (OIDC)** providers such as Login with Amazon, Facebook, and Google, and **Security Assertion Markup Language (SAML)** providers such as Shibboleth or Active Directory Federation Services. SPs include AWS services, social media platforms, and online banks.

AWS SERVICES THAT SUPPORT IDENTITY FEDERATION

- **AWS IAM** — enable a separate SAML 2.0 or OIDC IdP per account, pass user attributes (cost center, title, locale), and set fine-grained, attribute-based permissions.
- **AWS IAM Identity Center** (successor to AWS SSO) — centrally manage federated access and SSO across many accounts and business apps; works with IdPs such as Okta or Microsoft Entra ID (Azure AD) through SAML 2.0 and assigns access by IdP group membership.
- **AWS STS** — a web service that issues temporary AWS credentials and lets an IAM user, federated user, or application assume an IAM role.
- **Amazon Cognito** — add sign-up, sign-in, and access control to customer-facing web and mobile apps; scales to millions of users and supports social and SAML sign-in.

Workforce identities are the human users who are members of your organization (as opposed to application or service users). If those users can already authenticate — for example, by signing in to your corporate network — you can federate them into AWS using either IAM or IAM Identity Center rather than creating an IAM user for each person. At a high level: the user authenticates against a local directory with an ID and password, an outside system presents that authentication to IAM, IAM returns a **temporary credentials token** by way of AWS STS, and the user accesses the protected resources with those temporary credentials.

AWS STS is the engine behind temporary access. It is a web service (API) that provides **temporary, limited-privilege** credentials for IAM users, federated users, or applications. When the `AssumeRole` operation succeeds, STS returns credentials that typically last from a few minutes to several hours and are not recognized once they expire; a user with permission can request new ones before or after expiry. `AssumeRole` is the standard mechanism for **cross-account access and for federation**. An **identity broker** — an intermediary proxy that connects service providers with identity providers — sits between the external IdP and AWS: it authenticates the user against the IdP, requests temporary credentials from STS, and passes them to the application.

CONSOLE SSO VIA AN OIDC IDENTITY BROKER	CONSOLE SSO VIA SAML 2.0
corporate directory + identity broker	*the portal itself is the IdP*
▪ User opens an app and submits a user ID and password ▪ The **identity broker** checks the corporate identity store (Active Directory or LDAP) ▪ On success, the broker requests temporary AWS credentials from **AWS STS** ▪ The app receives the credentials and redirects the user to the AWS Management Console — an SSO experience with no separate AWS sign-in	▪ User navigates to an internal portal that acts as the SAML IdP ▪ The IdP authenticates against the identity store (LDAP or AD) ▪ The portal returns a **SAML assertion** to the client ▪ The client posts the assertion to the AWS SAML sign-in endpoint, which calls `AssumeRoleWithSAML` on STS and returns temporary credentials + a sign-in URL

Amazon Cognito is the preferred way to add federation to customer-facing applications. It is a fully managed service that provides **authentication, authorization, and user management** for web and mobile apps: users can sign in directly with a username and password or through a third party such as Apple, Facebook, Google, or Amazon, or through a SAML enterprise IdP. Cognito has two main components — **user pools** and **identity pools** — and the distinction between them is the single most tested idea in this section.

USER POOL – AUTHENTICATION	IDENTITY POOL – AUTHORIZATION
a directory and IdP; issues tokens	*exchanges tokens for AWS credentials*
▪ A user directory where users sign in or federate through a third-party IdP ▪ Collects user attributes into a **JSON Web Token (JWT)** issued to your app ▪ Can act as a standalone IdP, drawing on the OIDC standard ▪ Provides a hosted web UI for sign-up, sign-in, MFA, and password reset ▪ **User pool groups** manage permissions and represent user types (readers, contributors, editors)	▪ Creates unique identities and assigns permissions to users ▪ Exchanges user-pool tokens for **temporary AWS credentials** (using STS behind the scenes) ▪ Grants access to AWS services such as an S3 bucket or a DynamoDB table ▪ Can generate temporary credentials for **unauthenticated guest** users

TABLE 9.2 Amazon Cognito user pool features

FEATURE	WHAT IT PROVIDES
Sign-up	Let users create a profile native to the pool, redirect them to a third-party IdP, or create users from a data source
Sign-in	Use the pool as a standalone user directory and IdP for your app
Federate third-party identities	Handle the tokens returned from social sign-in (Facebook, Google, Amazon, Apple) and from OIDC and SAML IdPs
Hosted UI	Present customizable Cognito-hosted pages for sign-up, sign-in, MFA, and password reset
Support for JWTs	Use tokens to reach server-side resources or exchange them for temporary AWS credentials
User pool groups	Group users to manage permissions or represent different types of users

WORKED EXAMPLE – COGNITO AUTHENTICATE THEN AUTHORIZE

The goal is to authenticate a user and then let them reach another AWS service. First, an app user **signs in through a Cognito user pool** and, on success, receives **user pool tokens**. Next, the app **exchanges those tokens for AWS credentials through a Cognito identity pool**. Finally, the app uses those **temporary AWS credentials to access other AWS services** — the same two-step split you saw in the Cognito lab, where the user pool created and managed users for the Birds website and the identity pool granted access to the DynamoDB table.

COMMON PITFALL

User pool vs. identity pool is the classic Cognito trip-up. Use a **user pool** when you need to design sign-up/sign-in pages, manage user data, or run a custom authentication flow. Use an **identity pool** when users need **access to AWS resources** (an S3 bucket, a DynamoDB table) or when you need temporary credentials for guest users. Authentication is the user pool; authorization to AWS is the identity pool.

9.3 Managing access to multiple accounts

When AWS supports the different teams in an organization, there are two general patterns for isolating resources. The first defines **multiple VPCs in a single AWS account** — attractive when you want centralized information-security management with minimum overhead. The second creates **multiple AWS accounts, each with its own VPC**. In practice, both large and small organizations tend to create multiple accounts — separate accounts for business units, or for development, test, and production environments — often placing common resources such as DNS, directory services, and content management in one shared account.

ADVANTAGES OF MULTIPLE ACCOUNTS	CHALLENGES OF MULTIPLE ACCOUNTS
isolation and cost benefits	*the price of scale*
▪ Isolation by business unit or department ▪ Isolation by environment (dev, test, production) ▪ Isolation of auditing and recovery data ▪ Separation of accounts for regulated workloads ▪ Easy per-unit cost alerts ▪ Cost savings from bulk/volume pricing across accounts	▪ Managing security consistently across accounts ▪ Manual effort to create many new accounts ▪ Deciding which organization gets billed ▪ The need for centralized governance to ensure compliance and consistency

AWS Organizations is an account-management service that consolidates multiple AWS accounts into a single, centrally managed organization. It provides account creation and management, **consolidated billing** with **tiered pricing discounts**, and the ability to group accounts hierarchically into **organizational units (OUs)** — which can nest up to **five levels** deep. Its signature governance feature is the **service control policy (SCP)**, which specifies the maximum permissions for member accounts. Organizations builds on IAM by extending control to the account level: a user can do something only if **both** Organizations and IAM policies allow it, and if either one blocks the operation, it is denied.

> SETTING UP GOVERNANCE IN AWS ORGANIZATIONS
>
> - **Create a hierarchy of OUs** — the organization automatically creates a parent container called **root**; you define OUs beneath it.
> - **Assign accounts to OUs** as member accounts (for example, internal IT, engineering, development, production).
> - **Define SCPs** that apply permission restrictions to specific member accounts.
> - **Attach SCPs** to the root, an OU, or an account — the policy flows **outward from the root**, affecting everything beneath the attachment point.

An **SCP offers central control over the maximum available permissions** for the accounts in your organization: it governs which services and API actions are reachable by IAM users and roles in member accounts, and it affects the **entire account**. Critically, an SCP **grants no permissions** — it defines a **guardrail** that limits what an account's administrator can delegate. IAM policies defined in the individual accounts still apply, and a **local administrator cannot override** an SCP. The best practice is clear: it is far easier to define a policy once in an SCP than to replicate the same permission settings into IAM policy documents in every account.

COMMON SCP SCENARIOS

Typical uses of SCPs include **blocking a service or action** — for example, denying users the ability to disable AWS CloudTrail in all member accounts; **enforcing tagging** — for example, not allowing an EC2 AMI to launch unless it carries a required tag; and **preventing member accounts from leaving the organization**. That last one is a one-statement policy: an `Effect` of `Deny` on the `organizations:LeaveOrganization` action for `Resource` all, which stops any member account from leaving.

A **permissions boundary** is a related but distinct control. It is an advanced feature that uses a managed policy to set the **maximum permissions an identity-based policy can grant to a single IAM entity** (a user or role). Like an SCP, it grants nothing on its own — the entity can perform only the actions allowed by **both** its identity-based policies and its permissions boundary. For instance, if a boundary allows only `s3*`, `cloudwatch*`, and `ec2*`, then an identity-based policy granting `iam:CreateUser` **fails**, because IAM is not within the boundary — even with no explicit deny anywhere.

WORKED EXAMPLE – HOW MULTIPLE POLICY TYPES COMBINE

A user in a test OU is subject to three policies at once: an **SCP** on the OU that denies `ec2*` and `sqs*`; a **permissions boundary** that allows `s3*` and `sqs*`; and an **identity-based IAM policy** that allows `ec2:DescribeInstances`, `kms*`, `s3*`, and `sqs:SendMessage`. Work each request through all three layers — remembering that a request is allowed only where every layer allows it and that any explicit deny wins.

TABLE 9.3 Effective permissions for the test-OU user

REQUEST	ALLOWED?	WHY
`ec2:DescribeInstances`	No	The explicit `Deny` on `ec2*` in the SCP overrides the allow in the IAM policy
AWS KMS (`kms*`)	No	No explicit deny, but KMS is not within the permissions boundary
Amazon S3 (`s3*`)	Yes	Within the boundary, not denied by the SCP, and granted by the IAM policy
`sqs:SendMessage`	No	The explicit `Deny` in the SCP overrides the allow in the boundary and the IAM policy

PERMISSIONS BOUNDARY	ORGANIZATIONAL SCP
scopes one entity	*caps whole accounts*
▪ Applies to an IAM entity (user or role) ▪ Sets the max permissions its identity-based policies can grant ▪ Does not grant permissions ▪ Often used to explicitly allow a small subset of services	▪ Applies to all members of an organization or OU ▪ Sets the max permissions for account members ▪ Does not grant permissions ▪ Often used to deny a specific set of services (for example, deny Amazon RDS to the Internal IT OU)

To stand up and govern a multi-account environment without building all of this by hand, **AWS Control Tower** automates the setup and governance of a secure, well-architected multi-account environment. It establishes a **landing zone** — a multi-account baseline based on best-practices blueprints — and enforces **guardrails** (also called controls), which are high-level rules expressed in plain language for security, operations, and internal compliance. It is the

recommended path whether you are starting fresh on AWS or already run multiple accounts but want built-in blueprints and prescriptive guidance at scale.

SAMPLE EXAM QUESTION, WORKED

A company has two testing accounts — one for performance testing, one for integration testing — grouped into one AWS Organizations OU, each with a Tester role. Testers must be limited to Amazon EC2 and Amazon RDS, may only start and stop EC2 instances, and need read/write on RDS. Which TWO tasks implement this? The answers are **create an SCP** that limits the OU's available services to EC2 and RDS, and **create an IAM policy** that grants the specific EC2 and RDS permissions to the Tester role. The distractors fail on attachment rules: an SCP **cannot be attached to a role** and cannot be created in a member account, and an IAM policy **cannot be attached to an OU**. SCPs set the boundary; IAM policies grant the detail.

COMMON PITFALL

Remember what an SCP is and is not. It **grants nothing** — it only caps the maximum — and it attaches only to the **root, an OU, or an account**, never to an IAM user or role. Pair every SCP guardrail with IAM identity-based policies that actually grant the permissions users need.

9.4 Encrypting data at rest

Sensitive data must be protected both **in transit** (moving between networks) and **at rest** (residing on a local or cloud storage device). This section focuses on data at rest. Protecting it upholds the **confidentiality and integrity** of the **CIA triad** — confidentiality keeps data hidden from unauthorized parties, integrity ensures it is not altered, and availability keeps it reachable by the right people. Encrypting data at rest makes it far harder for an attacker to compromise data **even if they compromise an endpoint**, and it is often required for business or compliance reasons.

Encryption uses a code called a **cipher** — which contains algorithms to both encrypt and decrypt — to turn readable **plaintext** into unreadable **ciphertext** that only the right **key** can reverse. A key is a series of numbers and letters the algorithm uses. A strong algorithm produces ciphertext that cannot be decrypted with any practical amount of computing power without the key, which is precisely why **protecting and managing keys becomes the critical part of any encryption solution**. Encryption comes in two types — symmetric and asymmetric — and both can protect data in transit or at rest. **Envelope encryption** combines the two, as the TLS (SSL) protocol does.

SYMMETRIC ENCRYPTION	ASYMMETRIC ENCRYPTION
one shared key	*a public/private key pair*
▪ The **same key** encrypts and decrypts — a shared secret between sender and receiver ▪ Typically **faster** and efficient for large amounts of data ▪ Widely used and generally accepted as secure; change the key frequently ▪ Use it when speed, cost, and low overhead matter, when encrypting large volumes, or when the data stays inside the network	▪ A **public key** encrypts and a **private key** decrypts; every party has a key pair ▪ More secure but **slower** — longer keys and more complex math ▪ Can provide **non-repudiation** — the sender cannot deny sending ▪ Use it when sharing data outside the org, when regulations forbid sharing the key, or when segregating key access by role

ENVELOPE ENCRYPTION

- Encrypt the item with a key (the **data key**).
- Encrypt that data key with another key (a **key-encryption key**).
- Continue wrapping each key inside another key to the desired number of layers.
- Store the encrypted key alongside the encrypted item — like locking a safe key inside a safety-deposit box, then locking that key away in turn.

CLIENT-SIDE ENCRYPTION (CSE)	SERVER-SIDE ENCRYPTION (SSE)
you encrypt before it reaches AWS ▪ The application encrypts data **locally before** sending it to AWS and decrypts after receiving it ▪ You **create and manage your own keys** ▪ Keys and algorithms are known only to you ▪ Supported by libraries such as the AWS Encryption SDK, the DynamoDB Encryption Client, and S3 encryption clients	*AWS encrypts on your behalf* ▪ AWS encrypts data **at its destination** after receiving it ▪ Services **transparently** encrypt on write and decrypt on access ▪ Keys can be managed by AWS ▪ Most AWS services offer or default to SSE, integrated with AWS KMS

The two approaches are **not exclusive** — you can apply CSE and SSE to the same data for a stronger security profile. With SSE and Amazon S3, for example, you can upload data over HTTPS and the service endpoint handles encryption and key management for you before writing to disk. That leads to the managed key service at the center of most AWS encryption: **AWS Key Management Service**.

AWS KMS is a managed service that lets you create and control the cryptographic keys used to protect your data. It uses **FIPS 140-2 validated hardware security modules (HSMs)** to protect keys, integrates with many other AWS services, and lets you set **usage policies** that determine which users can use each key. You can create keys with aliases, **automatically rotate** them on a schedule, disable or delete them, and even **import your own** keys. The keys **never leave KMS** — you submit data to KMS to be encrypted or decrypted under keys you control — which greatly reduces the risk of a key being compromised. Every request is logged in **AWS CloudTrail**, so you can audit who used which key and when.

TABLE 9.4 AWS KMS key types

KEY TYPE	MANAGED BY	NOTES
Customer managed key	You	KMS keys in your account that you create, own, and manage
AWS managed key	An AWS service, on your behalf	In your account, created and used for you by an integrated service
AWS owned key	An AWS service, across many accounts	Not in your account; protects your resources but is not visible to you
Data key (symmetric)	You, outside KMS	Returned to you to encrypt data locally, including large data and other keys
Data key pair (asymmetric)	You, outside KMS	A related public/private key pair; the private key never leaves KMS unencrypted

KMS CRYPTOGRAPHIC OPERATIONS

- `Encrypt` — encrypt plaintext of up to **4,096 bytes** under a KMS key (small secrets such as a password or identifier).
- `Decrypt` — decrypt ciphertext produced by KMS operations; for an asymmetric key, you also specify the algorithm.
- `GenerateDataKey` — return a symmetric data key: a plaintext copy plus a copy encrypted under a KMS key.
- `GenerateDataKeyPair` — return an asymmetric data key pair for use outside KMS.
- KMS keys are the **primary resource**; because they stay inside KMS, you must call KMS to use one in any cryptographic operation.

Many services integrate with KMS — including **Amazon S3** and **Amazon EBS** — so you can protect the data a service stores for you with keys in your account. One rule matters for the exam: **AWS services integrated with KMS use symmetric encryption KMS keys only; they do not support asymmetric KMS keys.** With S3, KMS provides server-side encryption of objects at the object level. With EBS, attaching an encrypted volume means the data at rest, the disk I/O, and any snapshots are all encrypted transparently on the servers that host the EC2 instance; snapshots of encrypted volumes — and volumes created from them — are automatically encrypted.

WORKED EXAMPLE – SSE-KMS WITH AMAZON S3 AND EBS

S3 upload (encrypt): you request to store a file as an encrypted object → S3 requests a data key from KMS → KMS generates a plaintext data key and a copy encrypted under your customer managed key, sending both to S3 → S3 encrypts the object with the plaintext key, stores the object, deletes the plaintext key, and keeps the **encrypted** key in the object metadata. **S3 read (decrypt):** you request the object → S3 sends the encrypted data key to KMS → KMS decrypts it with the customer managed key (which never leaves KMS) and returns the plaintext key → S3 decrypts the object, lets you open it, and deletes the plaintext key. **EBS:** each volume is encrypted with **AES-256-XTS** (two 256-bit keys); EBS is given a grant to your customer managed key, obtains an encrypted data key, and the decrypted data key is held in memory to encrypt and decrypt all traffic to and from the attached volume.

COMMON PITFALL

Two KMS facts commonly get twisted. First, KMS keys **never leave the service** — data comes to KMS, not the other way around — which is why detaching an encrypted EBS volume and disabling its key affects access. Second, integrated services encrypt your data with **symmetric** KMS keys only; if a question pairs S3 or EBS with an *asymmetric* KMS key, it is wrong.

9.5 AWS security services for securing user, application, and data access

A core part of the AWS Well-Architected Framework's **Security pillar** is applying security **at all layers** — a **defense-in-depth** approach. Limiting access, the focus of earlier sections, is one layer; others include keeping out unwanted traffic, adding data-protection mechanisms, and automating detection and response. AWS groups its purpose-built Security, Identity, and Compliance services into categories so you can reach for the right tool per use case, reducing the burden of writing your own security code.

TABLE 9.5 AWS Security, Identity, and Compliance categories

CATEGORY	PURPOSE	EXAMPLE SERVICES
Identity and access management	Securely manage identities, resources, and permissions at scale	IAM, IAM Identity Center, Amazon Cognito, AWS Organizations
Detection and response	Strengthen posture and streamline security operations	AWS CloudTrail, Amazon Detective, Amazon Inspector, AWS Security Hub
Network and application protection	Enforce fine-grained policies at network control points	AWS Network Firewall, AWS Shield, AWS WAF
Data protection	Protect data, accounts, and workloads from unauthorized access	AWS KMS, AWS Secrets Manager, Amazon Macie
Compliance	See compliance status and monitor continuously with automated checks	AWS Artifact, AWS Audit Manager

TABLE 9.6 Defense-in-depth service spotlight

SERVICE	WHAT IT DOES	EXAMPLE USE CASE
AWS WAF	Web application firewall that monitors HTTP/HTTPS requests; managed or custom rules; allow/block by IP, country, or headers; includes AWS Shield for DDoS at no extra cost	Block requests missing a User-Agent header; stop malicious account sign-ups
Amazon Macie	Discovers and classifies sensitive data (PII) in Amazon S3 using ML and pattern matching; surfaces data-security risks	Flag sensitive data being migrated into S3 for admin review
Amazon Inspector	Vulnerability management that continuously scans EC2 instances, ECR container images, and Lambda functions for software flaws and network exposure	Scan EC2 AMIs and report findings before deployment
Amazon Detective	Analyzes and investigates the root cause of findings using ML, statistics, and graph theory; up to a year of data; links to GuardDuty	Triage an issue by finding all activity for a specific IAM entity
AWS Security Hub	Aggregates findings across accounts, services, and third-party tools; checks against FSBP and other frameworks; automation rules; security score	Prioritize response by correlating and aggregating diverse findings

AWS Trusted Advisor rounds out the picture. It is not security-specific, but it inspects your account and makes recommendations across **five categories** of AWS best practices — cost optimization, security, fault tolerance, service limits, and performance improvement. It is reachable through the AWS Management Console and available to **all support tiers**: Basic and Developer support get core security checks and all service-quota checks, while Business and Enterprise support get every check. After you enable **AWS Security Hub**, its security controls and findings also appear in the Trusted Advisor console — a single place to watch your posture.

CHOOSING A SERVICE BY USE CASE

- Guard the application edge and absorb DDoS — **AWS WAF** with **AWS Shield**.
- Find and classify sensitive data sitting in Amazon S3 — **Amazon Macie**.
- Continuously scan EC2, containers, and Lambda for vulnerabilities — **Amazon Inspector**.
- Investigate and find the root cause of suspicious activity — **Amazon Detective**.
- Consolidate findings and monitor overall posture against best practices — **AWS Security Hub**, whose recommendations also surface in **Trusted Advisor**.

COMMON PITFALL

Match the service to the verb in the scenario. **Macie** *discovers* sensitive data — and only in **Amazon S3**. **Inspector** *finds vulnerabilities* in compute (EC2, ECR images, Lambda). **Detective** *investigates root cause*. **Security Hub** *aggregates* findings from services like Macie and Inspector. Swapping any of these is a common distractor.

Module summary

- Attaching a policy to every user does not scale — use IAM groups (one policy, many members, built around job functions) and remember groups cannot nest and hold no credentials of their own.
- Scale further with ABAC (permissions matched on tags in a single policy) instead of RBAC (policies that list each resource and must be edited as the environment grows).
- Across identity policies, group policies, permissions boundaries, and SCPs, a request is allowed only where all applicable types allow it — and an explicit deny in any one overrides every allow.
- Identity federation is a trust between an IdP (authenticates) and an SP (controls resources); AWS STS issues temporary, limited-privilege credentials via AssumeRole and AssumeRoleWithSAML, often brokered by an identity broker.
- Amazon Cognito adds auth to web and mobile apps: user pools authenticate and issue JWT tokens; identity pools exchange those tokens for temporary AWS credentials to reach AWS services.
- Most organizations run multiple accounts; AWS Organizations consolidates them, offers tiered-pricing consolidated billing, and groups accounts into OUs (nested up to five levels).
- SCPs set the maximum permissions (a guardrail) for member accounts and grant nothing; they attach to the root, an OU, or an account — never to a role — and must be paired with IAM policies that actually grant access. AWS Control Tower automates a governed multi-account landing zone.
- Encrypt data at rest for confidentiality and integrity: symmetric (one fast shared key for large data), asymmetric (a slower, more secure public/private pair), and envelope encryption (wrap the data key under another key).
- CSE encrypts before data reaches AWS with keys only you hold; SSE encrypts transparently at the destination, usually with keys managed by AWS KMS, whose FIPS 140-2 HSM-backed keys never leave the service and are audited in CloudTrail.
- AWS provides defense-in-depth security services by category — WAF/Shield at the edge, Macie for sensitive data in S3, Inspector for vulnerabilities, Detective for investigation, Security Hub to aggregate findings, and Trusted Advisor for best-practice recommendations.

Review questions

1. **Why do IAM groups scale better than per-user policies, and what two things can a group never do?**

   Answer: One policy attached to a group is inherited by every member, so a single edit updates everyone instead of one user at a time. A group can never be nested inside another group, and a group has no security credentials of its own and cannot call AWS directly — it only organizes user permissions.

2. **When would you choose ABAC over RBAC, and what plays the role of an attribute in AWS?**

   Answer: Choose ABAC when users and resources grow rapidly: one attribute-based policy grants access wherever tags on the identity and resource match, so new users and resources need no policy edits. In AWS, the attributes are tags (key or key-value pairs) applied to IAM users and roles and to AWS resources.

3. **In Amazon Cognito, what is the difference between a user pool and an identity pool?**

   Answer: A user pool is a user directory and IdP that authenticates users (directly or via a third party) and issues JWT tokens. An identity pool takes those tokens and exchanges them (through STS) for temporary AWS credentials so the app can access AWS services such as S3 or DynamoDB; it can also grant guest access.

4. **What does AWS STS return, and which operations are used for federation and cross-account access?**

   Answer: STS returns temporary, limited-privilege credentials that last from minutes to a few hours and stop working when they expire. AssumeRole is used for cross-account access and general federation, and AssumeRoleWithSAML is used when a SAML assertion drives the sign-in.

5. **How do an SCP and a permissions boundary differ, and what do they have in common?**

   Answer: An SCP applies to all member accounts of an organization or OU and attaches to the root, an OU, or an account; a permissions boundary applies to a single IAM entity (user or role). Both set a maximum — a guardrail — and neither grants any permissions on its own.

6. **A user has an SCP denying ec2 and sqs, a permissions boundary allowing s3 and sqs, and an IAM policy allowing ec2:DescribeInstances, kms, s3, and sqs:SendMessage. Which of these can the user do?**

   Answer: Only Amazon S3. EC2 and SQS are explicitly denied by the SCP (a deny overrides any allow); KMS is not denied but falls outside the permissions boundary; S3 is within the boundary, not denied by the SCP, and granted by the IAM policy.

7. **Contrast client-side and server-side encryption, and explain envelope encryption.**

   Answer: With CSE the application encrypts data locally before sending it to AWS and manages its own keys; with SSE the receiving service encrypts data transparently at its destination, usually with AWS-managed keys. Envelope encryption encrypts the data with a data key, then encrypts that data key under another key, storing the encrypted key with the data.

8. **Which security service fits each need: sensitive data sitting in S3, vulnerabilities in EC2 and Lambda, root-cause investigation, and aggregating findings across accounts?**

   Answer: Amazon Macie discovers sensitive data in Amazon S3; Amazon Inspector scans EC2, ECR container images, and Lambda for vulnerabilities; Amazon Detective analyzes and investigates the root cause of findings; and AWS Security Hub aggregates findings across accounts, services, and third-party products.