Introduction to cybersecurity

SIO

deti universidade de aveiro
departamento de eletrónica, telecomunicações e informática

João Paulo Barraca

The imagery associated with hackers, hooded figures in dark rooms, scrolling green terminals, dramatic masks, comes almost entirely from film and television rather than from practice. It persists because a story needs a visible villain, and because the visual vocabulary is convenient shorthand. The consequence is that most people, including managers who approve security budgets, hold a mental model of the threat that is both inaccurate and, more importantly, misaligned with where their real risk sits.

The terminal windows in that kind of imagery usually list real offensive tools, things such as port scanners, packet capture utilities, exploitation frameworks and password crackers. Those tools genuinely exist and are genuinely used, but they are used openly by defenders too: the same scanner is how a security team discovers its own unpatched servers. Nothing about the tool itself is illegal; what separates a professional from a criminal is written authorisation, a scope, and rules of engagement.

The economics of real cybercrime explain why the stereotype misleads. Crime is organised as a business, with ransomware-as-a-service platforms that rent malware to affiliates in exchange for a share of the ransom, dedicated negotiators who talk to victims, infrastructure sold as a subscription, and even HR-style support channels for affiliates. Attacks are chosen because they are cheap, scalable and monetisable: phishing, credential theft, ransomware and business email compromise, not cinematic real-time duels with an intruder.

The defensive reality is closer to ordinary office work: analysts reading alerts and tickets, engineers reviewing code and configurations, auditors checking that access reviews happened, and staff writing and enforcing policies. That matters pedagogically because the skills required are largely about systematic method, documentation, communication, and persistence. A useful question to carry through the course is this: if your answer to 'is this cybersecurity?' only covers attackers and software, what has your definition left out? Paper records, door locks, hiring decisions, supplier contracts, and health-and-safety style accidents all belong somewhere in the field.

Cybersecurity

Subject focused on the predictability of systems, processes, environments…

- Across all aspects of a (business, system, organization) life cycle:
  - Planning
  - Development
  - Execution and operations
  - Processes
  - Human resources and clients
  - Supply Chain
  - Mechanisms and Controls
  - Standards, Compliance and Laws, …

The organising concept here is predictability, which is a stronger and more useful idea than 'safe' or 'protected'. Think of any system as having a set of legitimate states and a set of legitimate transitions between them: a bank account balance and the transfers that change it, an authenticated session and how it may be used, an industrial valve and the commands that open it. A system is predictable when every state it can end up in, and every way it gets there, was anticipated by its designers. A security failure is then simply the system arriving at a state nobody planned for.

That framing deliberately erases the boundary between attack, accident and natural failure. A flooded data centre, a disk that develops bad sectors, an administrator who mistypes a command, and an intruder who steals a credential all produce the same observable outcome: an unanticipated state. This is why resilience engineering, safety engineering and cybersecurity share techniques such as redundancy, checklists, rollback, least privilege, and post-incident reviews. Asking 'what states can this system be driven into, and by what?' is a single question with four very different sources of answer.

Listing the whole life cycle, from planning and development through operations, people, supply chain, controls and law, is a claim about where work must be done. A flaw introduced in planning is usually fixed on a whiteboard at almost no cost; the same flaw discovered during development costs rework; found in production it costs an emergency patch, an outage, or disclosure; and after an incident it can cost litigation and regulatory penalty. The industry shorthand for this is 'shift left' or 'security by design'. Some flaws cannot be fixed late at all: if the supplier you depend on is untrusted, or the hire you made is untrustworthy, no later technical control fully compensates.

The lifecycle list is not arbitrary either. Management-system standards are organised in essentially the same way, which is why they feel familiar once you have seen this framing. ISO 27001, for instance, requires an organisation to understand its context and leadership commitment, to assess and treat risk, to plan and implement controls, to measure and audit performance, and to review and improve continually. Each stage named here maps onto a clause of that logic, which is one reason the standard has been so widely adopted as a way to structure a security programme rather than merely a checklist of technologies.

Security application use case: Planning and Design

Design of a solution complying with some requirements under a normative context

- Create solutions without flaws, not vulnerable to fraud and abuse
  - All possible operation states are the ones predicted
  - There are no additional states escaping the expected logic
    - Even if forced transitions are attempted
- Under the scope of a normative context
  - Specific for each activity or sector
  - Ex: ISO 27001, ISO 27007, ISO 37001, RGPD, PCI-DSS

Doing security work at the planning stage means producing an explicit account of what the solution is supposed to do and refuse to do, before anyone builds it. In practice this takes the form of structured threat modelling: enumerate the assets, the entry points, the trust boundaries, the actors, then ask systematically what could go wrong at each one. Common techniques include STRIDE, which asks in turn about Spoofing, Tampering, Repudiation, Information disclosure, Denial of service and Elevation of privilege, and abuse or misuse cases, which are written like user stories but describe an attacker abusing a feature rather than a user benefiting from it.

The phrase 'states escaping the expected logic' deserves a concrete example, because it is where most design-level vulnerabilities live. Consider a password reset flow: is the reset token random and single-use, or derived from the user's email address and timestamp? Consider an internal API that the mobile app trusts to have been called only after authentication: what stops a client from calling it directly? Consider an approval workflow where the same user can be requester and approver. None of these are coding mistakes; they are design decisions in which an unanticipated state was quietly allowed, and each is far cheaper to fix while the diagram is still a diagram.

The normative context matters because 'secure' is not a single global target: it is defined relative to rules. A management-system standard such as ISO 27001 specifies what a security programme must look like and is the only one of these that an organisation can be certified against. ISO 27002 gives detailed guidance on individual controls. An auditing guideline such as ISO 27007 describes how to audit a management system. ISO 37001 covers anti-bribery programmes, which shows that the 'normative context' idea is not limited to information security. The GDPR is a law, with regulators and fines; PCI-DSS is a contractual scheme enforced through payment contracts rather than through legislation. Each has a different enforcement mechanism, a different auditor, and a different consequence for failure.

One practical consequence: the same system designed for two sectors can legitimately have different security requirements. A payment system handling card data inherits strict contractual rules about storage, network segmentation and testing frequency. A hospital system inherits rules about patient records and about keeping life-relevant services available. A marketplace inherits consumer-protection and anti-fraud duties. Designing 'secure' without first asking 'secure with respect to which rules, and verified by whom?' produces either over-engineering or, far more often, a design that passes technical review and fails an audit.

Security application use case: Development

Implement a solution complying with the design, without other operation modes

- Without bugs or malicious code compromising the correct execution
  - No crashes
  - Without invalid or unexpected results
  - With the correct execution times
  - With adequate resource consumption
  - Without information leaks
    - Even side channels
- Software:
  - Requires careful implementation and dependency management
  - Requires tests to develop an implementation with the expected… and only the expected behavior

Development turns design intent into something executable, and it is where most of the historically exploitable flaws have actually been created. The classic family is memory corruption: writing beyond the bounds of a buffer, using memory after it has been freed, trusting a length field from untrusted input. For decades this accounted for a large share of the most severe vulnerabilities in browsers, servers and kernels, and it is why memory-safe languages are frequently discussed as a security measure rather than merely a productivity one: they make a whole category of mistake impossible instead of merely unlikely.

The list of requirements here is broader than 'no crashes' in an instructive way. Correct execution times and adequate resource consumption are security properties, because an unbounded loop or an unbounded query is a self-inflicted denial of service, and because algorithmic complexity attacks deliberately feed inputs chosen to make a hash table or a sort degrade from fast to quadratic. Information leaks include the ones a developer never intended at all: verbose error pages that disclose framework versions and file paths, stack traces returned to clients, debug endpoints left enabled, and logs that record passwords or tokens in plaintext.

Side channels are the subtlest entry on the list because they involve no logic error. If a login comparison stops at the first mismatched character, the response is measurably faster for a wrong first character than for a correct one, and an attacker can reconstruct a secret one character at a time purely by timing; that is why secret comparisons are written to be constant-time. Hardware itself leaks: the Spectre and Meltdown families of attacks showed that a process can infer data it should not be able to read by observing how fast the processor's speculative execution behaves. Power consumption, cache occupancy, electromagnetic emanations and even fan noise have all been used to infer secrets.

Dependency management is arguably the defining development-security problem of the last decade. In many commercial codebases, code written by the organisation is a minority of what ships; the remainder is open-source libraries, containers, build tools and transitive dependencies pulled in automatically. A single widely-used component can therefore become a single point of failure for the whole industry, as with the Log4Shell incident in a mainstream Java logging library, which exposed hundreds of thousands of systems almost overnight, or the compromised maintainer-style attacks where an attacker takes over a small popular package and injects malicious code into it. Defences are administrative as much as technical: pinning versions, generating a software bill of materials so you can answer 'do we use it?' in minutes, monitoring advisories, and treating third-party code as untrusted input rather than as free infrastructure.

Security application use case: People and Supply Chain

Staff and Partner actions cannot compromise states

- Internal staff
  - Internal resources can also be considered as attackers
    - Malicious developers that introduce bugs and flaws
    - Resources that exfiltrate internal property or client data
  - Internal resources are popular victims of malicious attackers
    - Mostly through social engineering
    - Once compromised, they enable actors to access confidential assets
  - Selection of collaborators, Training, Awareness, Support, Tools and processes as ways to prevent internal attacks
- Solution providers
  - External providers are frequently exploited to reach larger organizations
  - Their tools, their personnel, their resources
  - Companies must carefully build and monitor their suppliers for potential attacks.
  - External Staff is trained to distinguish correct from incorrect behavior

Treat internal staff as potential attackers is uncomfortable but standard practice, and it splits into three quite different risks. Malicious insiders intend harm: a dismissed engineer who deletes production data, a developer who plants a logic bomb, a support agent who sells customer records. Negligent insiders mean no harm but cause it anyway: shared credentials, personal cloud storage for confidential files, a database left reachable from the internet. Compromised insiders are simply victims: their laptop or account has been taken over by malware or a phishing campaign, and the attacker now walks around with legitimate credentials. The three require different mitigations, and treating them as one problem is a common cause of a badly designed programme.

For compromised and negligent insiders, the effective measures are technical constraints plus training: least privilege so a stolen account cannot reach everything, logging and monitoring so anomalous access is visible, separation of duties so no single person can complete a sensitive process alone, sensible defaults, and enforced offboarding that revokes access on the last day. For malicious insiders, the emphasis shifts to detection and to limiting the blast radius, since prevention is weak: hiring checks, referral and behaviour signals, and controls such as two-person rules for high-impact operations. Organisations should also be careful about employee privacy and proportionality, since mass surveillance of staff is legally constrained in many jurisdictions, including under EU data-protection law.

Social engineering is how internal people become entry points, and it works because it targets the human incentives rather than the software: urgency from a supposed executive, flattery from a fake supplier, fear from an apparent IT or police authority, curiosity from an attachment named payroll. Modern campaigns are increasingly targeted, using LinkedIn and company websites to invent plausible roles and conversations, and increasingly assisted by synthetic voice and video for fake video-call authorisations. Measuring awareness is done with simulated phishing, and the useful metric is not how few people click but how many people report, since a reported attempt is a defensive win.

Supplier compromise is now one of the highest-leverage attacks available, because it converts one breach into many. Attacking a widely-deployed software vendor can plant a backdoor in a product trusted by thousands of organisations simultaneously, an update mechanism can be abused to deliver malicious code with all the legitimacy of a signed release, and a contractor with remote administrative access to a large enterprise is a shortcut to that enterprise's core systems. Defences include supplier due-diligence before contracting, contractual security and audit clauses, software bills of materials, signature verification and code signing, network segmentation of supplier access, just-in-time and time-boxed vendor credentials, and continuous monitoring of suppliers rather than one-off assessment. The underlying lesson is the same one as the rest of this slide: trust is transitive!

Security application use case: Processes

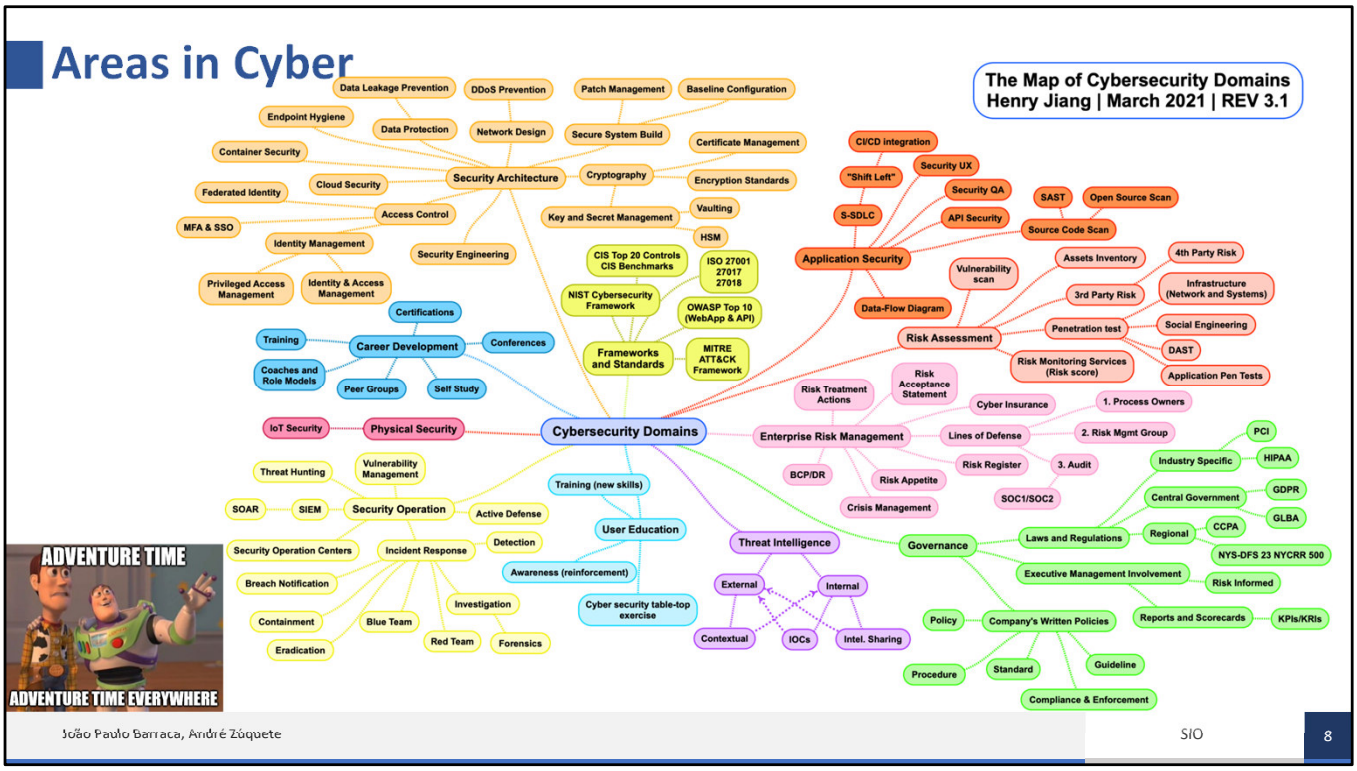
ISO 27001 – Clean Desk Policy

A clean desk policy is a small, concrete administrative control with a large pedagogical value: it requires almost no technology and depends entirely on repeated human behaviour. In practice it typically states that documents containing sensitive information are stored in locked furniture when unattended, that print jobs are collected immediately, that whiteboards are erased, that removable media is locked away, that screens are angled away from visitor sightlines and locked when idle, and that material is shredded rather than discarded in ordinary waste. Auditors check it by walking through offices outside working hours and looking.

The reason such a policy exists in information-security standards is that information lives on paper, on screens, on whiteboards, on packaging, and on removable media, and none of that is protected by a firewall. An attacker who needs a system password may simply photograph a keyboard or a laminated card near one; an attacker who needs a name badge may follow an employee through a door rather than forge one. Physical access frequently upgrades into full system access, because unattended machines are often already authenticated, and network sockets in a meeting room are usually trusted.

Process controls of this kind share three properties worth remembering. They are cheap to write and expensive to sustain, because compliance decays the moment enforcement stops feeling real. They are auditable, which is why standards prefer them, but audits sample rather than cover, so they can pass while the everyday practice is poor. And they work best when paired with a technical mechanism that removes the need for vigilance: automatic screen locking, locked cabinets as standard furniture, badge-controlled printing that releases jobs only at the machine, and no printers at all in high-sensitivity areas. The general rule: use technology to remove the burden of a habit, and use policy for the residual behaviour that technology cannot cover.

The other entries visible here, covering visitor escorting, clear-screen rules, identification badges and secure disposal, all address the same underlying exposure: casual, low-tech exposure of sensitive material. They also illustrate that security controls are frequently inherited from one another across domains; the physical-security practice of escorting visitors is conceptually identical to the digital practice of not letting unverified third-party software run with local privileges. Once you see the pattern, you can predict most controls without memorising them.



The areas of cybersecurity are not just a diagram, they are how the industry divides labour, hires people, and writes exams. Organisational security concerns policy, risk management and compliance, typically staffed by governance specialists. Physical security concerns buildings, guards, badges, cameras and environmental systems in data centres, often shared with facilities management rather than with IT. Information security concerns the data itself, classification, handling rules and cryptographic protection. System and network security concerns hosts, networks and their configuration, usually owned by operations teams. Operational security concerns the day-to-day running: patching, backups, monitoring, incident response. Secure development concerns the software that a business builds and buys.

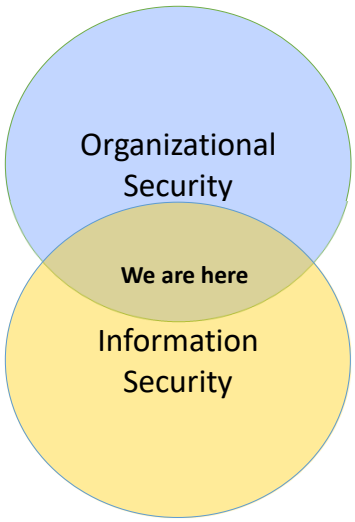
Who owns each area varies enormously between organisations, and that ambiguity is itself a source of risk. A lost laptop touches physical security (the device), information security (the data it held), operational security (whether it was encrypted and whether backups existed) and organisational security (whether a mobile-device policy existed and was followed). If four departments each believe one of the others owns that risk, nothing gets done, which is a common finding in incident post-mortems and in audits of immature programmes.

The taxonomy is also incomplete in a way worth flagging, because new areas keep being carved out as markets mature: cloud security, operational technology and industrial control system security for factories and utilities, mobile and application security, privacy engineering, identity and access management, threat intelligence, and cryptography engineering. Most practitioners identify with one or two of these areas while needing working knowledge of all of them, and this course deliberately covers the shared vocabulary first, precisely so that later, deeper material in any one area can be read without relearning the basics.

Security Domains or Areas

Security is scoped into domains with many overlaps

- **Organizational Security**
- Physical Security
- **Information Security**
- System Security
- Operational Security
- Secure Development



It is worth being able to state each domain in a sentence and give one control for it. Organisational security: the framework of policy, risk management, roles and compliance that holds everything together, for example a documented risk register reviewed quarterly. Physical security: protection of premises and equipment, for example badge-controlled doors and CCTV. Information security: protection of data wherever it lives, for example classification with handling rules and encryption. System security: protection of hardware, software and networks, for example hardened configurations and network segmentation. Operational security: protection through correct daily running, for example patching, backups, monitoring and incident response. Secure development: protection built into how software is designed, coded, tested and released, for example code review and dependency management.

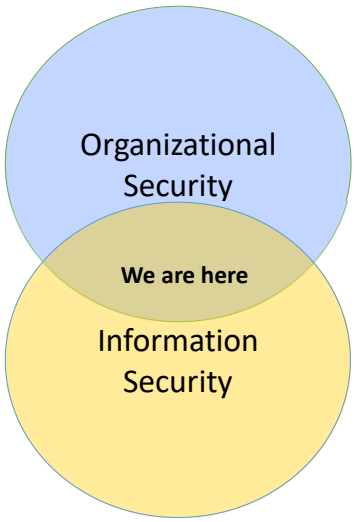
The overlaps are deliberate rather than sloppy, and they carry a practical warning: no real incident respects the boundaries. A ransomware event involves secure development if the initial entry was an insecure web application, system security if lateral movement exploited flat networks, operational security if backups were untested, information security if the stolen data was over-classified or over-retained, and organisational security if the incident-response plan did not exist or notification duties were missed. Analysing an incident across all domains is normally much more informative than analysing it inside one.

The visual navigation on this kind of slide also models something real about how professionals specialise. Certifications behave the same way: broad management-oriented credentials concentrate on organisational and information security, technical credentials concentrate on system, network and operational security, and development credentials concentrate on secure development, with the result that two certified professionals can use the same word, for instance 'control', and mean slightly different things. Keeping the six domains distinct in your head, while remembering they overlap in practice, is a useful discipline for the rest of the course.

Security Domains

Organizational Security (ISO 27001)

- Measures to **protect data** (electronic and otherwise) collected, held, and processed,
- and to protect its **computer systems**, **devices**, **infrastructure**, **computing environment**, **information and data stored** and all **other relevant equipment**
- from damage and **threats** whether internal, external, **deliberate, or accidental**.



The definition quoted here is intentionally broad in two ways. It does not distinguish digital from physical information: paper contracts in a filing cabinet, a printed list of client accounts, a whiteboard in a meeting room, and a production database all fall inside the same scope, because the object of protection is information and the medium is incidental. And it does not distinguish causes: internal or external, deliberate or accidental. A fire, a flood, a stolen laptop, a bored administrator, and a criminal crew are all treated as threat sources against the same asset.

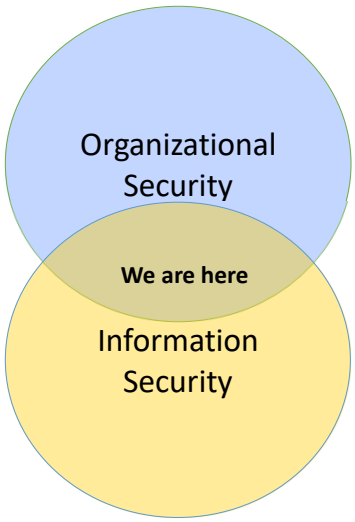
That second choice is what makes a single, unified risk register possible. Instead of running three registers, one for crime, one for accidents and one for natural hazards, an organisation asks one question per asset: what could stop this being confidential, intact or available, how likely is it, and what would it cost? That is what lets a control decision be made consistently across causes, and it is why the same evaluation methodology covers both 'attacker exploits our VPN' and 'air-conditioning failure in the server room'.

The wording also hints at the scope of an information security management system, which is the structure ISO 27001 asks organisations to build around this definition: define the scope of the systems and information covered, assign roles and responsibilities, assess risks to the assets in scope, decide which risks to treat, mitigate, transfer or accept, choose and document controls, then measure, audit, review and improve. Certifiers audit that management system rather than the technical state of the estate, which surprises people: an organisation can be certified and still get breached, because the certificate attests that the organisation manages security, not that it is impenetrable.

Security Domains

Information Security (ISO 27001)

- preservation of **confidentiality**, **integrity**, and **availability** (CIA) of information.



- **Confidentiality**: Ensuring that information is accessed only by authorized individuals.
- **Integrity**: Maintaining the accuracy and completeness of information.
- **Availability**: Ensuring that information is accessible when needed by authorized users.

The three properties are best understood operationally, as questions you can ask about any system. Confidentiality asks who can see this, and expects a named, enforced answer. Integrity asks whether this is correct and complete, and expects mechanisms that detect or prevent unauthorised change. Availability asks whether the people who need it can get it when they need it, and expects numbers: how quickly must the system recover, how much data may be lost, and what happens when a component fails.

They interact in ways that produce most real design trade-offs. Encrypting data protects confidentiality, but losing the key destroys availability permanently and can silently destroy integrity if you cannot tell a corrupted ciphertext from a wrong key. Backups protect availability and integrity, but a backup set containing everything is a large confidentiality liability, and ransomware increasingly targets backups first. Making a system highly available tends to mean more exposed services, more replication paths and more administrative access, all of which enlarge the confidentiality and integrity attack surface. Security engineering is largely the job of deciding which of the three your business most cannot afford to lose, and choosing the least damaging compromise.

The triad is minimal on purpose, and there are well-known extensions. Authenticity asks whether you are talking to who you think you are. Accountability or non-repudiation asks whether an action can be attributed later, which is why signed audit trails matter in regulated settings. Utility asks whether the information is actually usable, and retention asks whether you kept it as long as required, or no longer than permitted. The value of the original three is that they remain the shared vocabulary of essentially every framework, so knowing them precisely lets you read a standard, an audit report or a vendor whitepaper without translation.

Information Security Objectives

- **Confidentiality**: Ensuring that information is accessed only by authorized individuals.
- Measures:
  - Encrypt information
  - Use access passwords (strong)
  - Use Identity Management and Authentication systems
  - Doors, Strong walls
  - Security personel
  - Training

Encryption is the first measure on the list, and its practical difficulty is almost never the mathematics. Transport encryption protects data in transit, encryption at rest protects stored data, and end-to-end encryption protects content from the infrastructure that carries it. What fails in practice is key management: keys stored beside the data they protect, no rotation so a key leaked years ago still works, backups encrypted with a key stored in a wiki, or no key escrow at all so the data becomes permanently unreadable when an employee leaves. Any discussion of confidentiality measures is really a discussion about who holds which keys, and under what controls.

Passwords are the weakest commonly-used authentication factor, and their weakness is human rather than algorithmic: people reuse them, they persist after breaches, and they can be guessed or phished. Strong password policy means length over complexity, uniqueness per service, and password managers to make that feasible. Multi-factor authentication removes most remote password risk in one step, which is why it is often the first high-value control recommended. Identity management goes further, centralising authentication on a single provider and deriving authorisation from roles or attributes, so that disabling one account truly disables access everywhere.

One subtle failure deserves mentioning: metadata. Content encrypted properly can still leak through its size, its timing, its frequency, its sender and recipient names, and its access patterns. Email headers reveal social networks even when bodies are encrypted. File names in a encrypted backup reveal project names. Query patterns to a search service can reveal a topic of investigation. And devices that are decommissioned, resold or returned, including disks, phones, printers and even USB tokens, regularly turn up with recoverable data, which is why secure disposal, cryptographic erasure and asset-decommission procedures belong in any confidentiality discussion.

Information Security Objectives

- **Integrity**: Maintaining the accuracy and completeness of information.

- Measures:
  - Identity control (hashes)
  - Backups
  - Access controls
  - Robust storage devices
  - Data verification processes

A hash is a fingerprint of data: a short value computed from the content, such that changing the content changes the fingerprint with overwhelming probability. Hashes are therefore excellent detectors: package managers verify downloads, filesystems and backup tools detect silent corruption, git uses content hashes as object identifiers so tampering with history is visible, and intrusion-detection tools compare hashes of system binaries against known-good values. What a plain hash cannot do is prove who produced the data, or prevent the change in the first place.

Two extensions fix that. A keyed message authentication code proves that a message with a given key was produced by someone who knows the key, so a receiver knows both integrity and origin. A digital signature does the same with asymmetric keys and adds non-repudiation, which is why software updates, firmware images, code-signing for operating systems, document signing, and certificate chains depend on it. Note the recurring practical failure mode: the cryptography is fine, but the signing key is stolen, or the verification step is skipped for convenience, or unsigned legacy artifacts are allowed. Integrity depends on the discipline around the mechanism, not on the algorithm.

Integrity is also why access control and logging are listed together with hashes. A hash tells you data changed; an access-control system limited who could change it; a log tells you who did and when. Removing the ability to change, recording the attempts, and detecting the changes are three complementary controls, and mature systems add a fourth, recovery: known-good copies from which you can restore. That is the correct place to understand backups here: they are not only about restoring availability after a disaster, but about restoring correctness after corruption, whether the corruption was caused by a bug, a mistake or an attacker.

Finally, note that 'accuracy and completeness' has a business dimension too. Duplicate records, silently truncated fields, unit mismatches, timezones, partial imports, and unreviewed manual edits quietly corrupt operational data with no attacker involved, and the consequences can be severe in finance, health and industry. Verification processes, reconciliation between systems, dual control or four-eyes approval on sensitive changes, schema constraints, and validation on input are treated as security controls for that reason, and they are cheap relative to the errors they prevent.

Information Security Objectives

- **Availability**: Ensuring that information is accessible when needed by authorized users.

- Measures:
  - Backups
  - Disaster recovery plans
  - Redundancy
  - Virtualization
  - Monitoring

Availability is the property businesses notice fastest, and therefore the one they can articulate most easily. It is normally expressed through numbers rather than adjectives: a service-level agreement stating the maximum downtime permitted per month, a recovery time objective stating how quickly service must be back, and a recovery point objective stating how much data loss in time terms is tolerable. Setting those numbers is a business decision that technical teams then have to design against, and a recovery point objective of 'no loss' has very different cost from 'last night's backup'.

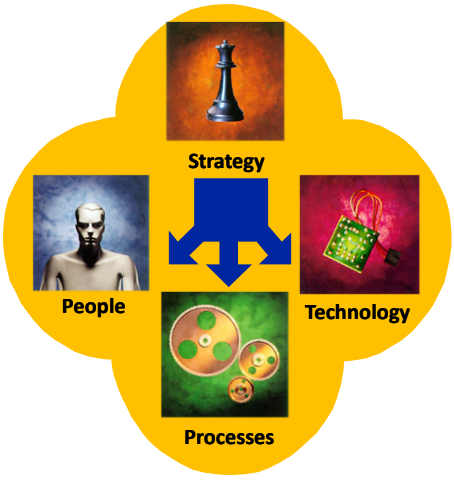
Redundancy and disaster recovery solve different problems and are frequently confused. Redundancy, duplicated power supplies, disks in RAID, clustered servers, load balancers, multiple network paths, data centres in separate regions, aims to prevent the outage from being visible at all; it usually costs money continuously and requires failover to be tested, because untested failover routinely fails when needed. Disaster recovery assumes the outage happens, and is a documented, rehearsed plan: who declares it, in what order services come back, how authentication and connectivity are restored first, where people work from, and how long it truly takes to rebuild from backups, which is often measured in days rather than minutes.

Attackers target availability precisely because it requires the least privilege. Volumetric denial of service floods a link; protocol attacks exhaust stateful resources such as connection tables; application-layer attacks send a small number of expensive requests, for example a search or a report generation, and take down a service with a fraction of the traffic. Amplification and reflection techniques let a small input generate a much larger flow towards the victim. Defences include capacity headroom, content delivery and anycast networks that absorb flooding near its source, scrubbing services, rate limiting, timeouts and resource quotas, and queueing so a service degrades gracefully instead of collapsing.

A large share of real-world outages are self-inflicted: a bad deployment, a certificate that expires, an exhausted connection pool, a lockout policy that locks out administrators, a storage volume filled by verbose logging, a DNS change, a misconfigured autoscaling rule. That is why monitoring and alerting appear in the list alongside backups, and why operational practices such as staged rollouts, rollback plans, capacity review, configuration change management, and drills matter as much as hardware. Backups are the last line here, and their practical requirements are that they are automatic, verified by real restore tests, retained long enough to cover the time an attacker sits unnoticed, and protected so that compromising a production administrator cannot delete them. Availability work also carries a confidentiality and integrity cost: every redundancy path is another place data is copied, and every administrator able to trigger failover is another privileged account to protect.

How can we use security in an organization?

With a strategy following the organizational dimensions



- Selection
- Training
- Awareness
- Organization of security

- Vulnerability scanning
- Firewalls
- Authentication
- Access Control
- Cryptography
- Digital Signatures
- Certification authorities
- Certification hierarchies
- etc...

- Security policies
- Security administration processes
- Continued evolution of auditing and follow-up processes

The division into strategy, people, processes and technology is a way of saying that a security programme is a management system, not a product purchase. Strategy decides what matters, what risk appetite is, and where money goes. People execute, and are also the largest attack surface. Processes make execution repeatable and auditable, and define who does what when something goes wrong. Technology enforces what humans cannot reliably remember to do.

The classic failure pattern is heavy investment in technology with thin investment in the other three. A next-generation firewall, endpoint detection tooling and a monitoring platform bought with no trained analysts watching them produces alerts that nobody triages; a documented incident-response plan that nobody has rehearsed collapses on contact with a real incident; a strong password policy that no one can comply with produces shared accounts written on paper. Buying is fast, visible and politically safe; changing behaviour and rewriting process is slow and harder to demonstrate, which is precisely why budgets drift towards products.

The strategy dimension is also where security meets the rest of the business. Security objectives should be derived from what the organisation is trying to do, its regulatory obligations, and what it cannot afford to lose, rather than from a list of technologies. A common way to express this is a security roadmap tied to maturity levels: identify a small number of high-value improvements, implement them, measure them, then move on, rather than attempting everything at once and completing little. The next slides make the argument that an imbalance between these dimensions is itself the vulnerability attackers find.

Notice also the recursive relationship between the four: strategy is expressed through policies, policies need processes to be followed, processes rely on people and are supported by technology, and technology decisions are only as good as the process that configures and monitors them. Auditors exploit this recursion, asking not only whether a control exists but whether someone is accountable for it, whether there is evidence that it worked, and whether anyone reviewed it recently.

Pitfalls

Pushing one dimension without the other weakens the security posture

- What may have failed?
  - Technology?
  - Processes?
  - People?
  - Strategy?
- Walls, firewalls, processes… everything bypassed

The image here summarises a pattern visible in an enormous number of actual breach narratives: a technically strong perimeter, defeated by one ordinary-looking email to one person who was never trained to be suspicious of it. Attackers are economically rational. They do not attack the strongest control; they search for the cheapest way in, which is very often a credential handed over willingly, a supplier with weaker defences, an unpatched forgotten system, or a business process that requires an exception.

When such an event is reviewed, the root cause is usually a missing or unbalanced dimension rather than a missing product. If people were not trained, the corrective action is training and reporting channels, not another appliance. If a process required an exception, for instance a temporary administrator account that was never removed, the corrective action is an access review process. If the strategy tolerated an asset being exposed for convenience, the corrective action is a decision about the asset itself. Audits and post-incident reviews that end with 'we bought something' are frequently the sign that the real cause was left untouched.

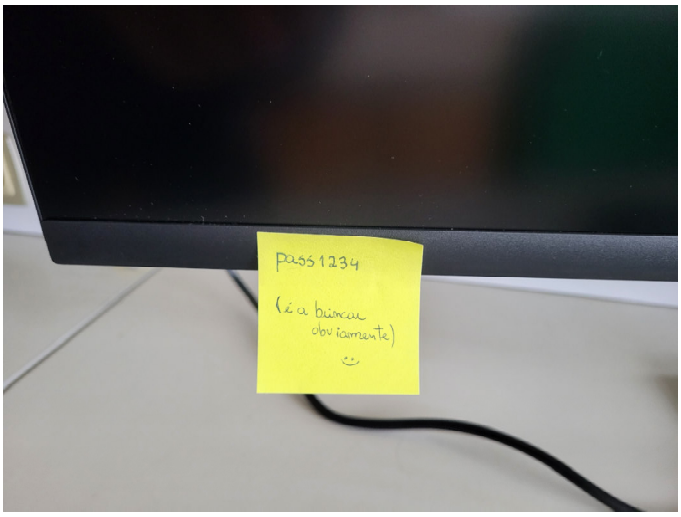
The phrase 'everything bypassed' is worth taking seriously as a design principle rather than only as a scare story. If a single control failure can defeat every other control, the architecture has a single point of failure, and mature design asks the opposite question: which controls would still work if this one failed? That question leads directly to defence in depth, segmentation, least privilege, monitoring, and recovery plans, all of which appear later in this material.

There is also an accountability lesson: the employee who clicked is rarely the responsible party in a well-run organisation. A culture that punishes the clicker stops the reporting that defenders most need, and the useful organisational question is why the attack succeeded against that process, that control, and that training, and who owned the decision.

Pitfalls

Pushing one dimension without the other

- When it works and there is a security culture, it’s part of the business.



A security culture is present when the desired behaviour happens when nobody is watching, and when people report problems rather than hide them. Concrete markers include an employee reporting an unusual email within minutes, a developer refusing to merge code without a security review even under schedule pressure, a manager declining an exception request that would weaken a control, and staff asking 'what does this mean for the data?' before starting a new project.

Cultures are built by a small number of consistent practices rather than by posters. Leadership behaviour is the strongest signal, because employees watch whether executives are exempt from rules. Making the secure path the easy path matters more than making warnings louder: single sign-on, default encryption, sensible templates, and pre-approved cloud configurations all remove the incentive to work around controls. Blame-free reporting, where near-misses are studied rather than punished, is what turns the population into sensors. And feedback matters: people stop reporting if reports vanish without acknowledgement.

Culture can be measured, imperfectly, and the measurements chosen shape the outcome. Click rates on simulated phishing tell you about susceptibility; report rates tell you about engagement, and the second is usually the better indicator. Other proxies include how many security issues are raised by staff before an audit finds them, how quickly incidents are reported by users rather than detected by tools, how many exceptions and exemptions are outstanding, and how many projects complete a threat model voluntarily. A programme where everyone knows that 'security is everyone's job' is often, in reality, a programme where nobody is accountable, so culture and clear ownership must be built together.

The business case is easier to make in a mature culture than in a control-heavy one, because controls that depend on human vigilance degrade silently. A technology control keeps working while powered on; a process control drifts. That is why the mature combination is a few strong technical mechanisms, clear ownership, and a workforce that notices and reports when those mechanisms behave strangely.

Security objectives

1/3 – Intrinsic and unavoidable aspects

- Defense against catastrophic events
  - Natural phenomena
  - Abnormal temperature, lightning, thunder, flooding, radiation, …
- Degradation of computer hardware
  - Failure of power supplies
  - Bad sectors in disks
  - Bit errors in RAM cells or SSD, etc.

This category exists to make a point that is easy to miss: a large share of the work called cybersecurity has nothing to do with adversaries. Physical and environmental causes kill and disable systems constantly. Data centres are protected against floods by siting decisions and drainage, against fire by detection and suppression systems that must be safe around electronics, against power loss by uninterruptible supplies and generators with tested fuel contracts, against overheating by climate control with redundant cooling, and against earthquakes by seismic bracing of racks. Surge protection matters because lightning-induced voltage spikes destroy equipment and corrupt storage.

Hardware degradation is a related and quantifiable phenomenon, and it is why operations teams plan for failure rather than assume correctness. Disks develop bad sectors and their failure rates rise sharply with age; consumer-grade media is not rated for the write volumes of modern servers; RAM cells flip bits, which is why critical machines use error-correcting memory and why large-scale computing infrastructure measures cosmic-ray-induced bit flips as a normal operating condition; fans, power supplies and network cards wear out. Availability controls, redundancy, health monitoring, replacement cycles, and firmware updates, are security controls in the sense used earlier, because they keep the system inside its predicted set of states.

The practical implication for a security programme is scope. If your risk register covers only attackers, you will underinvest in environmental and hardware resilience, and you will be surprised by an outage that has nothing to do with a crime. This is also why standards such as ISO 27001 include controls for equipment protection, supporting utilities, cabling, and environmental conditions: they treat these as part of the same management system, and auditors expect to see them documented and tested.

A final subtlety is that these causes interact with security controls. A flood can disable a backup site. A power event can corrupt a database just as badly as an attacker. A hardware fault can silently disable logging, which is discovered only when someone checks that logging is alive. Designing for intrinsic and unavoidable failure means asking not only 'can this fail?' but 'will I notice, and can I still recover?'.

Security objectives

2/3 – Unpredictable ordinary failures

- Defense against ordinary faults / failures
  - Power outages
  - Systems' internal failures
    - Linux Kernel panic, Windows blue screen, OS X panic
    - Deadlocks
    - Abnormal resource usage
  - Software faults
  - Communication faults...

Ordinary failures are distinguished by source rather than consequence: they come from the complexity of the system itself. An operating system kernel panic, a blue screen, a system that deadlocks because two processes hold each other's locks, memory exhausted by a leak, a filesystem that fills up, a service that restarts in a loop, a network link that flaps, a certificate that expires, a DNS record that was never updated, and a race condition that appears only under load. None requires malice, and none is exotic.

What makes these failures security-relevant is the same logic as before: they drive a system into states that were not predicted. A database that fails mid-transaction and recovers inconsistently has lost integrity. An authentication service that fails open because of a timeout has lost confidentiality. A monitoring pipeline that silently stops has lost the ability to detect anything at all, which is often described as the most dangerous failure of all, because it removes the evidence that would reveal other failures.

This is why reliability engineering and security engineering converge on shared techniques. Checklists, staged rollouts, rollback, canary deployments, health checks, timeouts and retries with backoff, circuit breakers, capacity limits, graceful degradation, chaos engineering in which failures are injected deliberately to discover unanticipated states, and blameless post-incident reviews all serve both disciplines. The phrase used in industry is that resilience is a security property, because an attacker's most reliable lever is often to push an ordinary failure mode harder than normal operations ever did.

There is a design lesson in the list too: prefer explicit handling of the failure case to implicit assumptions about the normal case. If a component can only ever be exercised under good conditions during testing, its failure behaviour is effectively untested, and it will be encountered for the first time during the worst possible moment. That is why the list of failure modes belongs in the design documents alongside the list of features.

Security objectives

3/3 – Threats

- Defense against non-authorized activities (adversaries)
  - Initiated by someone “from outside”, “from inside” or “through a supplier”
- Types of non-authorized activities:
  - Information access
  - Information alteration
  - Resource usage
    - CPU, memory, print, network, wallets, etc…
  - Denial of Service
  - Vandalism
  - Interference with the normal system behavior without any benefit for the attacker

Threats are the category with an adversary behind it, and the list here is a compact taxonomy of what adversaries actually do. Gaining information access covers eavesdropping, credential theft, scraping, insider exfiltration, and reading backups. Alteration covers tampering with data, configurations, firmware or logs, including the common tactic of editing audit logs to hide a trail. Resource usage covers taking control of computing capacity for the attacker's own purposes. Denial of service degrades availability. Vandalism and interference damage for their own sake.

Resource usage including 'wallets' is a direct reference to cryptomining abuse, where attackers install mining software on someone else's hardware and profit from the electricity and compute without stealing anything at all. It is discovered through unusual CPU or GPU load, outbound connections to mining pools, or unexpectedly high cloud bills. Related abuses include using compromised hosts as proxies for other crimes, as mail relays for spam, as nodes in botnets for denial-of-service, and as staging hosts for further intrusions. The important point for detection is that the malicious activity looks like legitimate compute rather than like data leaving the network.

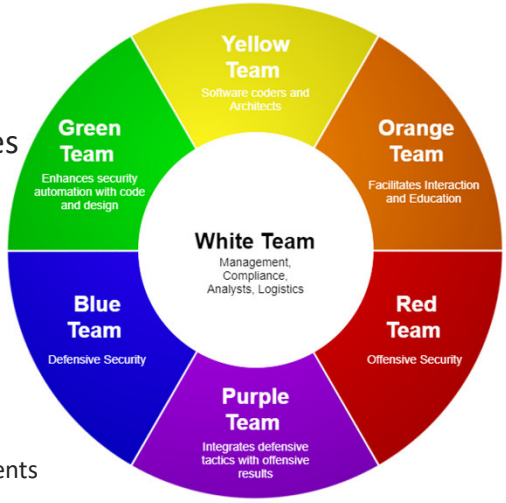
The origin of a threat matters as much as its type. External attackers may be opportunistic criminals scanning the whole internet, organised groups that target specific industries, or actors motivated by geopolitics. Internal attackers may be employees, contractors, or anyone whose credentials were obtained. Suppliers sit in a third position, and are attractive because they are usually weaker and trusted. A well-formed threat description names the actor, their capability, their motivation, and the asset they want, since that combination determines which controls are worth the money.

Vandalism deserves separate mention because it breaks the economic reasoning defenders often rely on. Much of defensive planning assumes attackers act to maximise profit and will stop when costs exceed gains. Ideological, retaliatory, or reputational attacks do not behave that way, and neither does an intruder who has already been paid a ransom and asks for more. This is one reason recovery planning exists alongside prevention: some adversaries cannot be deterred, only detected, contained, and recovered from.

Security Perspectives

Which type of approaches

- Defensive tasks: focus on maintaining predictability and building layers
  - Deployment of Firewalls, Backups, Alert systems
  - Creation of processes and compliance
- Offensive: focus on exploiting vulnerabilities in entities
  - May have malicious/criminal intent
  - May have the purpose of validating the solution (Red Teams)
- Other:
  - Reverse Engineering: Recovery of design from built products
  - Forensics: extract information and reconstruct previous events
  - Disaster Recovery: minimize the impact of attacks
  - Auditing: validate the solution complies with some set of requirements



The defensive and offensive sides of the profession are usually described with colours, and it is worth knowing the vocabulary because recruiters and vendors use it constantly. Red is the attacker role, authorised to find weaknesses by attempting them. Blue is the defender role, operating detection, monitoring and response with the tools the organisation has. Purple is not a third team but a deliberate collaboration between the two, in which red's findings are turned into blue's improvements on the spot, so that knowledge does not die inside a report nobody reads.

Three further colours describe the people who build things. Yellow is the builder role, developers and engineers who construct what red will attack and blue will defend. Green sits between yellow and blue, translating defensive knowledge into better build and configuration practices. Orange sits between yellow and red, teaching builders to think like attackers so that security is designed in rather than retrofitted. A white role referees engagements, defining and policing the rules of engagement, which matters because the same technical actions are criminal without authorisation and professional with it.

The 'other' column covers functions that support every colour rather than being a colour themselves. Reverse engineering recovers behaviour and design from products when source is unavailable, which is central to malware analysis, firmware inspection and interoperability. Forensics preserves and analyses evidence to reconstruct what happened, with attention to chain of custody because the output may be used in legal or disciplinary proceedings. Disaster recovery limits impact and restores operations. Auditing checks whether what is claimed is actually in place and effective, against a standard or a policy.

A practical note on offensive work: it is bounded by written scope, timing, prohibited techniques, and emergency contacts. Anything outside that authorisation is a crime, and professionals are expected to decline ambiguous engagements. Bug bounty programmes and coordinated vulnerability disclosure occupy the adjacent space where independent researchers report flaws to organisations, sometimes for payment, with agreed timelines before publication. Knowing where the legal line sits is part of the technical knowledge, not separate from it.

Core Concepts

1. Security Domains
2. Security Policies
3. Security Mechanisms
4. Security Controls

These four concepts form the spine of the vocabulary used by security management standards, and they nest inside one another. A domain is a boundary within which a set of rules applies, and which is under the authority of one trusted party. A policy is the set of rules that apply within a domain. A mechanism is the technical or physical means by which a policy is enforced. A control is the package of policy plus mechanism plus evidence, in a form that can be assigned, tested and audited.

The nesting matters more than the individual definitions, because it describes a chain from intent to proof. Someone decides what should be true, that decision is expressed as a policy for a named domain, the policy is implemented by one or more mechanisms, and the whole thing becomes a control once it has an owner, a defined scope, and evidence an assessor can inspect. Auditors walk this chain in reverse: given a claim that a risk is managed, which control addresses it, what policy defines it, what mechanism enforces it, and what evidence shows it worked last month.

Keeping the terms separate also prevents two common confusions. The first is treating a technology as a control: a firewall is a mechanism; a control is 'network traffic between the corporate network and the internet is filtered according to a documented and reviewed ruleset, with changes approved and logged', which includes the mechanism but also the process around it. The second is treating a document as security: a policy that no mechanism enforces and nobody verifies is an intention, not a control.

The rest of this section of the course takes the four concepts in order, and the distinction is worth holding on to because almost every later slide can be read as an example of it. When a term feels slippery, ask which of the four it is, and specifically who owns it and how anyone would know it is working.

Security Domains

A system or subsystem that is under the authority of a single trusted authority. Security domains may be organized (eg, hierarchically) to form larger domains.

- Allow managing security in an aggregated manner
  - Management will set the attributes of the domain
  - Entities are added do the domain and will get the “group” attributes
- Behavior and interactions are ruled by homogeneous rules inside the domain
- Domains can be organized in a flat of hierarchical manner
  - Flat: Domains do not overlap but have frontiers, and exist at the same abstraction level
  - Hierarchical: Domains have different levels of abstraction (Organization -> devices -> Servers -> ServerA)
- Interactions between domains are usually controlled
  - With gateways that limit, change or log interactions

Security domains exist to make management possible at all. Rather than writing rules for every individual user, device, process and data item, an administrator defines a domain, sets attributes once at the domain level, and adds entities to it so they inherit the attributes. This is aggregation, and it is why access control at any real scale is workable. It is also why domains must have clear boundaries: an entity that is not in any domain, or is in two with conflicting rules, is a management gap.

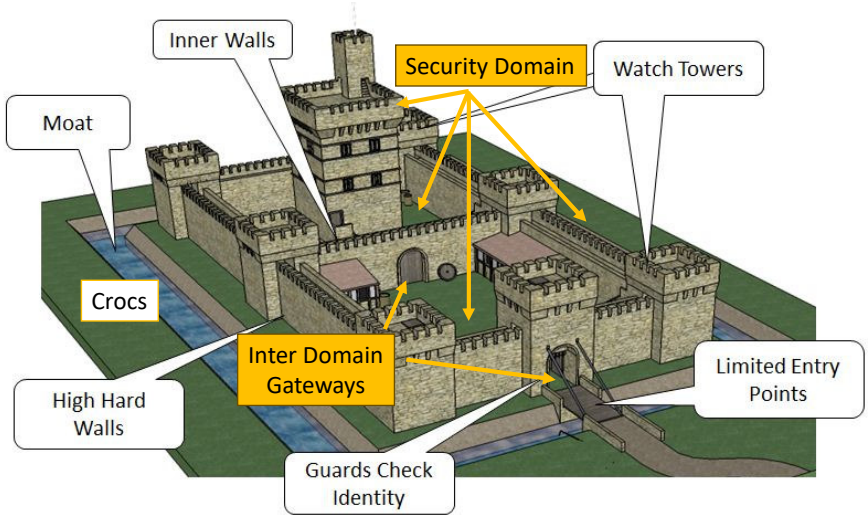
The flat versus hierarchical distinction describes how domains compose. In a flat arrangement, domains sit at the same level of abstraction and are separated by frontiers, as with network segments that partition a company by function. In a hierarchical arrangement, domains nest inside each other: the whole organisation, then regions, then server estates, then an individual server, then a single account on it. Nesting lets rules be inherited and overridden, which is exactly how directory services, cloud account structures, and role hierarchies behave, and it also creates the classic pitfall where an override at a lower level silently defeats a rule set higher up.

Gateways between domains are the interesting part. A gateway may limit interactions, for instance a firewall dropping traffic, transform them, for instance a proxy terminating TLS, a NAT rewriting addresses, or a transcoder stripping file formats, or record them, which is where the audit trail originates. Recording is easy to underappreciate and expensive to retrofit: without a log of what crossed a boundary and under which rule, neither an auditor verifying a control nor an investigator reconstructing an intrusion has anything to work with.

A modern wrinkle worth raising is that boundaries have become logical rather than physical. A domain today may be an identity boundary in a cloud directory, a virtual network, a tenant, or a policy boundary enforced by a service mesh rather than by a cable in a wall. That makes domains cheaper to create, and therefore easier to create carelessly; large organisations routinely end up with hundreds of domains, undocumented trust relationships between them, and orphaned accounts that still sit inside a trusted domain.

Security Domains

Popular application of security domains circa year 1500



The historical analogy works because a fortified settlement is a complete, legible implementation of the ideas being taught. The wall is a preventive control that stops casual entry. The moat is both preventive, it slows and obstructs, and detective, since crossing it is visible and audible. The gate is the gateway, the single controlled point where identity and purpose are checked, and it is also the strongest concentration of defenders, exactly as modern architectures concentrate inspection at chokepoints. The drawbridge is a gateway that can be removed entirely, which is a nice illustration of deny-by-default.

Layers are visible too: an outer ward, an inner ward, a keep. Each is a domain with its own gateway and its own authority. Once inside the outer ward, an attacker gains nothing like the access that reaching the keep would give, and must pass several independent controls. That is defence in depth in its oldest form, and it explains the strategic logic: rather than relying on one perfect boundary, trade space and time for exposure, so that progress is slow, visible, and interruptible.

The analogy also exposes weaknesses that map directly onto modern systems. A postern gate, a service entrance for supplies, is the unmanaged third-party connection. A water gate is the network path you forgot to secure because it was designed for legitimate traffic. Servants moving freely between wards are the service accounts with broad privileges. A traitor inside the walls is the insider threat that no wall addresses. Moats around only one side of the settlement are the partial scope of many real security programmes, and sieges that simply waited for starvation are denial-of-service and ransomware: attacks on patience, supply and continuity rather than on the wall.

Finally, notice the maintenance cost. Walls require constant repair, garrisons require training and pay, gates require rules that change with circumstances, and obsolete walls are expensive and get in the way of the city that grew around them. Legacy network perimeters behave exactly the same way, which is one motivation for the models discussed at the end of this session.

Security Policies

Set of guidelines related to security, that rule over a domain

- Organization will contain multiple policies
  - Applicable to each specific domain
  - They may overlap and have different scopes/abstraction levels
- The multiple policies must be coherent
- Examples
  - Users can only access web services
  - Subjects must be authenticated in order to enter the domain
  - Walls must be made of concrete
  - Communications must be encrypted

A policy is a rule that applies within a domain, and any real organisation has many of them, written at different levels of abstraction for different scopes. A high-level information security policy states principles and assigns accountability. Topic-specific policies cover access control, acceptable use, mobile devices, cryptography, supplier security, data classification, incident reporting, and remote working. Standards and procedures go further down, specifying concrete requirements such as minimum key lengths, password rules, or the steps for provisioning a privileged account.

Coherence between policies is the hard part, and it is a common audit finding. An HR policy that permits personal devices for work e-mail contradicts an IT standard requiring managed devices with disk encryption. A supplier contract promising rapid data access contradicts a rule that no external account may be created without two approvals. A marketing team's requirement for analytics contradicts the data-minimisation principle in the privacy policy. Contradictions of this kind survive for years because different departments own the documents and nobody is accountable for the intersection.

Because of that, policies are usually written technology-agnostically, deliberately: 'communications crossing an untrusted network must be protected cryptographically' rather than naming a protocol and version. This keeps the policy valid across technology changes and prevents the policy from being rewritten every time a product version changes. The trade-off is that an agnostic policy cannot be checked directly, so organisations attach a technical standard to it, which does name the protocol, and that standard is reviewed more often than the policy.

It is worth stating plainly that a policy without enforcement and verification is only a statement. So for every policy there should be an owner, a scope, at least one mechanism enforcing it, at least one measurement or review evidencing it, and a defined consequence for violation. When reading any policy document, look for those four things; their absence predicts the most common gap between 'documented security' and 'actual security', which auditors describe as an implementation failure rather than a documentation failure.

Security Policies

- Define the power of each subject
  - Least privilege principle: each subject should only have the privileges required for the fulfillment of his duties
- Define security procedures
  - Who does what in which circumstances
- Define the minimum security requirements of a domain
  - Security levels, Security Groups
  - Required authorization
    - And the related minimum authentication requirements (Strong/weak, single/multifactor, remote/face-to-face)

The power of a subject, meaning what that subject is permitted to do and to which objects, is the substance of access control. Two design choices dominate practice: assigning permissions to roles rather than to individuals, which makes rules reviewable and changes cheap when people move jobs, and deriving permissions from attributes such as department, clearance, device state or location, which is more flexible but much harder to reason about and audit. Most organisations use a mixture, and the practical difficulty is that role explosion quietly turns roles back into per-person exceptions.

Least privilege says each subject gets only the privileges needed for its duties, for the shortest time needed. Its value is measured in blast radius: a stolen account that can reach one application is a nuisance, one that can reach domain administration is an existential problem. It applies to software as much as to people, and service accounts are the classic violation, since they are created quickly, run unattended, are shared by several systems, are rarely reviewed, and are frequently given broad rights because that was the fastest way to make an integration work.

Keeping least privilege true over time is harder than establishing it. Access accumulates as employees change roles and old rights are never withdrawn, a phenomenon usually called privilege creep, and exceptions granted for emergencies become permanent. That is why the periodic access review, in which an owner confirms each person's rights or has them removed, is treated as a control in its own right, alongside joiner-mover-leaver procedures that automate provisioning and revocation, time-boxed privileged access granted on request with justification, and monitoring of privileged use.

The authentication strength requirement follows from the value of what is protected. Weak single-factor authentication may be acceptable for low-risk access; sensitive administration requires multiple factors, device-bound credentials, or hardware tokens; remote access requires more assurance than local access; and high-impact operations may require re-authentication plus a second person. Standards express this as a mapping from required authorisation level to allowed authentication methods, which is a useful document to write explicitly, because it stops authentication decisions being made inconsistently by whichever team builds a system.

Security Policies

- Define defense strategies and fight back tactics
  - Defensive architecture
  - Monitoring of critical activities or attack signs
  - Reaction against attacks or other abnormal scenarios

- Define what are legal and illegal activities
  - **Forbid list model**: Some activities are denied, the rest are allowed
  - **Permit list model**: Some activities are allowed, the rest is forbidden

Policies also decide how the organisation behaves when an incident happens, which is a very different question from how it prevents one. A defence architecture describes where controls sit and what each is expected to detect. A monitoring plan names the critical activities whose behaviour must be observed, meaning authentication failures, privileged use, configuration changes, unusual data volumes, and administrative access outside change windows. A response plan defines who is called, who decides containment, who talks externally, and what evidence must be preserved.

Fighting back is mostly containment, and containment choices carry cost and legal implications. Isolating a host may stop an intrusion but break a business process; disabling an account may stop credential misuse but alert the attacker that they have been detected; shutting a link may protect other systems while destroying evidence in flight. Reaching out to law enforcement, or actively attacking an adversary back, involves legal authority and jurisdictional questions, and 'hacking back' by victims is illegal in many countries. Real response plans therefore define decision rights and legal escalation, not just technical steps.

The permit and forbid list distinction is one of the most practically useful ideas in this session. A forbid list says certain things are prohibited and everything else is allowed, which is easy to adopt, tolerant of unanticipated legitimate use, and permanently reactive, since it only blocks what has already been identified as bad. A permit list says certain things are allowed and everything else is refused, which is stronger because unknown behaviour is blocked automatically, but requires knowing and maintaining what is legitimate, and requires an exception path so the business does not stall.

That distinction explains a lot of observed technology choices. Signature-based antivirus is a forbid list over known-bad code, which is why novel malware evades it; application allow-listing is a permit list over executables, which is why it is preferred on servers and industrial controllers with stable software. Default-deny firewall rules are a permit list; outbound-blocking blacklists are forbid lists. Least privilege and zero trust are permit-list philosophies. The general rule to take away: default-deny is stronger but more expensive operationally, and the exception process is where its security is won or lost in practice.

Security Mechanisms

- Mechanisms implement policies
  - Policies define, at a higher level, what needs to be done or exist
  - Mechanisms are used to deploy policies
- Generic security mechanisms
  - Confinement (sandboxing)
  - Authentication
  - Access control
  - Privileged Execution
  - Filtering
  - Logging
  - Auditing
  - Cryptographic algorithms
  - Cryptographic protocols

Policies say what must be true; mechanisms are how a system makes it true. The distinction is worth insisting on, because most security mistakes in practice are attempts to satisfy a policy with a mechanism that does not actually enforce it, for example a rule that 'only administrators may change server configuration' implemented on servers where everyone has local administrator rights because that was convenient for software installation.

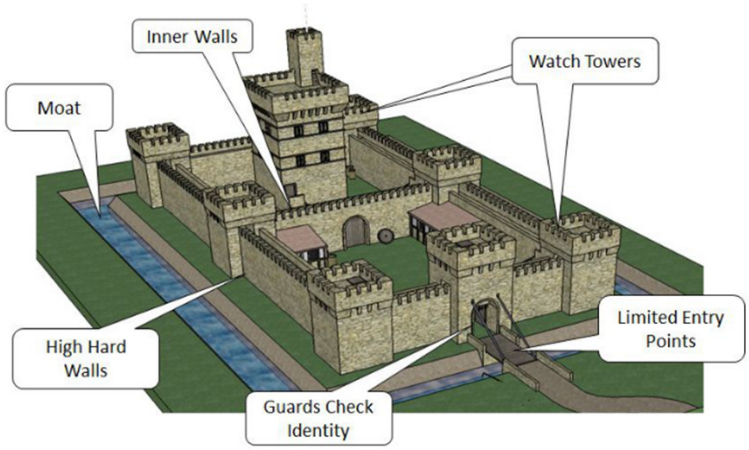
The generic mechanisms deserve a sentence each. Confinement or sandboxing restricts what a piece of code can touch, which is the same idea implemented by browser tab isolation, application containers, virtual machines, seccomp filters, and industrial controllers that only accept specific commands. Authentication establishes who or what a subject is, using something it knows, has, or is. Access control then decides what that subject may do. Privileged execution provides a controlled way to perform operations that ordinary subjects cannot, as with setuid programs, sudo, kernel mode, and hardware management engines. Filtering inspects and permits or drops content and traffic according to rules.

Logging records events so that behaviour can be reconstructed; auditing reviews those records, plus configuration and evidence, to check that policies were followed. The pair is what converts a control into something verifiable. Cryptographic algorithms provide the primitives, confidentiality, integrity, authentication, and signatures; cryptographic protocols turn them into usable systems, and protocol design is where most real failures occur, since the mathematics is rarely broken while the key management, downgrade handling and identity binding are routinely misused.

Two habits make this vocabulary useful rather than academic. First, when reading any design, ask which mechanism enforces each stated policy, and what happens if that mechanism is absent or fails. Second, remember that mechanisms compose badly: authentication plus access control plus logging is a coherent trio, but adding two sandboxes, an agent-based filter, and a proxy in a chain frequently produces behaviour nobody understands, performance nobody can explain, and gaps where each layer assumed the other was handling it.

Security Mechanisms

- **Policy**: Movement between domains is restricted
- **Mechanisms**: Doors, guards, passwords, objects/documents, training, salary



This example is deliberately chosen to break the habit of thinking of mechanisms as only technical. The same policy, that movement between domains is restricted, is implemented by doors and walls, by guards who check identity, by badges and passwords, by the rule that documents cannot leave a room, by training people not to escort strangers, and even by salary, since adequate pay and fair treatment reduce the incentive for insider fraud and are recognised in insider-threat literature as a real, if indirect, control.

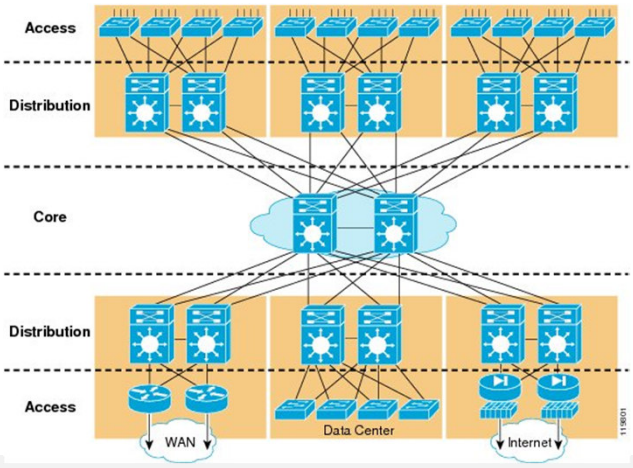
The multiplicity is not decoration. A door stops a determined-but-unsophisticated person; a guard can evaluate intent and refuse entry; a badge log provides evidence afterwards; a document rule reduces what can be carried out even if entry succeeds; training makes employees report the person who followed them in; and salary reduces the probability that someone inside cooperates voluntarily. Each mechanism fails differently, which is exactly what a layered design wants: an attacker strong against one is still blocked by another.

It is also worth noticing that several of these mechanisms are not preventive at all. Badge logs and guard reports are detective. Refresher training after an attempted intrusion is corrective. The physical door is preventive. Reading the same example through the prevention, detection and correction lens, which appears later, makes the vocabulary stick, and shows that one policy typically requires controls of all three kinds.

Finally, consider the cost side of the same example: doors and guards are expensive and slow the business down, badges and logs are relatively cheap and provide evidence, and training is cheap per person but unreliable alone. A realistic prevention argument would keep the cheap mechanisms broadly applied and reserve the expensive ones for the domains holding the most sensitive assets, which is precisely the risk-based allocation used throughout the rest of this material.

Security Mechanisms

- **Policy:** systems must be resilient to arbitrary failures of one component
- **Mechanisms:** equipment and links are doubled, protocols are developed



The second example shifts the same logic to resilience. The policy is that the system must survive the arbitrary failure of any single component, and the mechanisms are duplicated equipment and links, plus protocols designed to detect the failure and route around it. Note that the duplication alone is not the mechanism; the failover behaviour is. Redundant hardware with manual failover means an outage while somebody wakes up, diagnoses, and intervenes.

Failover design has a well-known set of difficulties. Detecting failure reliably is harder than it sounds, because a component that is slow is not obviously dead, and premature detection causes unnecessary switchover while late detection causes an outage. Split-brain situations arise when two sides each believe the other is dead and both take control, which can corrupt shared data; quorum and fencing exist specifically to prevent that. Failover capacity must be real, since a standby server too small for full load fails when it finally matters. And failover paths are usually untested, so they are the components most likely to be broken when needed.

There is a direct security consequence, which is why this belongs in a cybersecurity lecture rather than only in a reliability one. A failover configuration is a change to the effective security configuration, and standby paths frequently bypass controls: replication links left unencrypted, firewalls absent between primary and standby sites, temporary access opened for failover testing and never closed, or a disaster-recovery environment running older, unpatched software because it is rarely used. Auditors look for exactly this, and describe it as a second environment with a fraction of the protections.

The general principle from both examples is that a mechanism should be selected for the failure it needs to tolerate, and then tested under that failure. If you cannot describe the state in which the mechanism activates, nor demonstrate it activating, then its effectiveness is assumed rather than known, which is the definition of an unverified control.

Security Controls

A safeguard or countermeasure prescribed for an information system or an organization designed to protect the confidentiality, integrity, and availability

- Controls include policies & mechanisms, but also:
  - Standards and Laws
  - Processes
  - Techniques
- Controls are explicitly stated and can be auditable
  - E.g.: ISO 27001:2022 defines 93 controls across 4 themes
  - … organizational, people, physical, technological…

A control is the unit that accountability attaches to. It includes a policy and a mechanism, and also the standards, laws, processes and techniques that surround it, and its defining property is that it can be stated explicitly and audited. In audit language, a control has an owner, a scope, a stated purpose, an implementation, and evidence; if any of those are missing, an auditor records a finding even if the technical state is fine.

The ISO 27001 annex is the most widely referenced catalogue of controls, and the 2022 revision reduced it to 93 controls grouped under four themes: organisational, people, physical and technological. The flattening reflects a shift in the industry away from long granular lists towards attributes and outcomes, and it also acknowledges that many older entries were artefacts of 1990s technology, while newer controls cover things such as cloud services, protection of information for testing, logging and monitoring, and secure configuration of devices removed from the office.

The catalogue is a support, not the answer. Organisations implement the controls they need, and then document why the ones they skipped are not applicable, in a document usually called a statement of applicability. Certification assesses whether the management system identifies risks and covers them adequately, not whether all controls are present, so a small company with a well-argued subset can be certified while a large company with the full catalogue implemented badly cannot.

When comparing frameworks, it helps to know that the vocabulary differs while the substance overlaps. Control catalogues from other bodies, for instance the US National Institute of Standards and Technology's SP 800-53, the Center for Internet Security's hardening benchmarks and CIS Controls, or sector schemes such as PCI-DSS, all describe roughly the same set of safeguards organised differently, with crosswalks between them. Learning the concept of a control once lets you move between frameworks, which is a genuine professional skill since most practitioners end up reconciling two or three of them.

Types of Security Controls

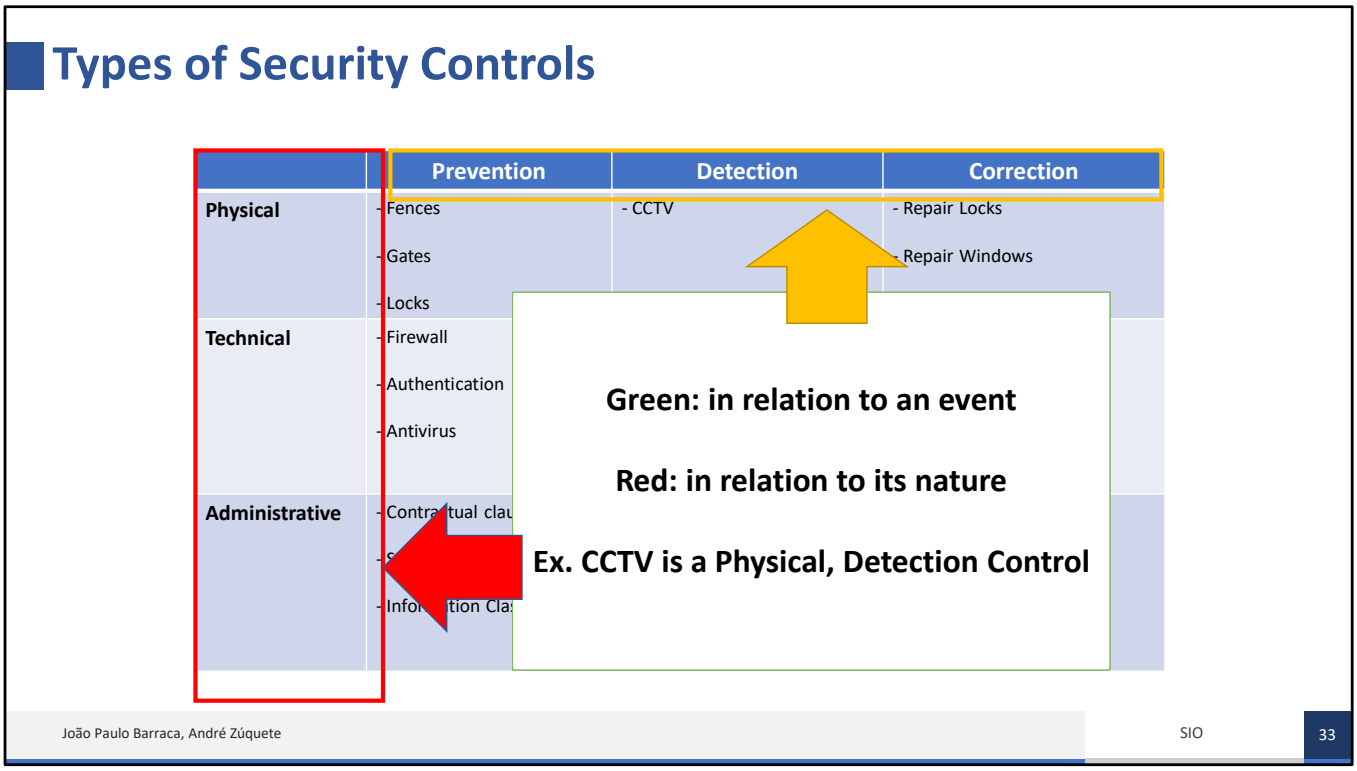
	Prevention	Detection	Correction
Physical	- Fences - Gates - Locks	- CCTV	- Repair Locks - Repair Windows - Redeploy access cards
Technical	- Firewall - Authentication - Antivirus	- Intrusion Detection Systems - Alarms - Honeypots	- Vulnerability patching - Reboot Systems - Redeploy VMs - Remove Virus
Administrative	- Contractual clauses - Separation of Duties - Information Classification	- Review Access Matrixes - Audits	- Implement a business continuity plan - Implement an incident response plan

The matrix sorts controls along two independent axes, and separating them is what makes the taxonomy useful. The temporal axis asks what the control does relative to an event: prevent it, detect it while or after it happens, or correct and restore afterwards. The nature axis asks what the control is made of: physical, technical, or administrative. Any control can be placed at an intersection, and most real risks are covered by several controls at different intersections.

The physical row shows how ordinary facilities measures behave. Fences, gates and locks prevent. Cameras detect, and importantly do not prevent anything on their own, which surprises people who expect a camera to deter and to stop. Repairing locks and windows, or reissuing access cards, corrects. The technical row is the familiar one: firewalls and authentication prevent, intrusion detection systems and honeypots detect, patching, rebooting, redeploying virtual machines and removing malware correct. The administrative row is the most neglected: contractual clauses, separation of duties and information classification prevent, access-matrix reviews and audits detect, and continuity and incident-response plans correct.

Two rows deserve extra attention in discussion. A honeypot is a detection control that deliberately offers something apparently valuable in order to observe interaction; it produces high-fidelity alerts because legitimate users have no reason to touch it, and it carries real risk, since it may attract attack and may be turned against the organisation. Separation of duties is a preventive administrative control with a strong track record in fraud, and it works by requiring two people for one sensitive outcome, which is also why it is resisted as 'slowing the business down'.

Finally, note that the administrative row is usually the cheapest to implement and the hardest to sustain, while the technical row is the reverse. Auditors consistently find that administrative controls are documented but not performed on schedule, for example access reviews that were due in March and happened in November, and that technical controls are performed but not reviewed, for example firewall rules accumulated over ten years with no owner. Both failure modes are visible directly on this matrix once you know where to look.



The annotation makes the two axes explicit: columns describe the control's relation to an event, rows describe its nature, and any single item can be described precisely by naming one from each, for instance CCTV as physical and detective. Being able to classify quickly matters because gaps become visible as empty regions, and because immature programmes are usually skewed: many preventive technical controls, few detective ones, and almost no rehearsed corrective ones.

The skew has a predictable consequence. With prevention only, the plan assumes the perimeter holds, and the first real incident arrives with no detection and no recovery path. With detection only, the organisation learns about every incident after damage and has invested nothing in reducing likelihood. Corrective controls are the least popular to fund because they pay for themselves only during a crisis, which is exactly when their absence is most expensive; buying them is the security equivalent of insurance, and it requires the same argument.

Completing the grid with familiar examples helps fix the categories. A badge reader is physical and preventive. A door that fails locked when power is lost is physical and preventive with an availability trade-off. A screen lock is technical and preventive. Log review is technical and detective. Restoring from backup is technical and corrective. A non-disclosure agreement is administrative and preventive. A quarterly access review is administrative and detective. A rehearsed incident-response plan is administrative and corrective.

A more advanced observation, worth raising with the class, is that controls frequently serve more than one temporal role at once, and this is a sign of design strength rather than of sloppy classification. Logging detects and also deters and supports recovery. Backup restores and also provides an alternate source for verifying integrity. An incident review prevents recurrence. When a control spans roles, it is usually worth more than a single-purpose control of similar cost, which is why good programmes deliberately look for those.

Practical security

Key concept: Realistic Prevention

- Consider that **perfect security is impossible**!
- Focus on the **most probable events** for the **most relevant assets**
  - May depend on physical location, legal framework, …
- Consider **cost** and **profit**
  - A great number of controls has a low cost
  - However, there is no upper limit on the cost of a security strategy
  - Security mechanisms must cost less than the asset it protects
- Consider all domains and entities
  - A single breach can be escalated to a more serious situation

Perfect security is impossible, and that is not a defeatist remark but an economic one: defence and attack are asymmetric, since the defender must be right everywhere and the attacker only once, and since defence costs money continuously while attack can be attempted repeatedly at low cost. Realistic prevention means accepting residual risk openly, and deciding what level of residual risk is acceptable, rather than pretending that the last few percent can be bought cheaply.

The prioritisation follows from asset value and probability. Identify the assets that would hurt most if lost, corrupted, or disclosed, then estimate plausible scenarios and their likelihood, and choose controls accordingly. The crude arithmetic behind this is annualised loss expectancy: expected loss per incident multiplied by expected frequency per year, compared against control cost, with the refinement that the control should also be evaluated for how much of the loss it actually removes, since a control that costs more than the risk it eliminates is a bad purchase even if technically sound.

The point about low-cost controls having no upper bound is worth unpacking: a single control is usually affordable, but security requirements multiply across assets, systems, suppliers and locations, so total programme cost grows without a natural ceiling. That forces ranking, and ranking requires a shared measure of asset value, which organisations frequently lack. Cost-effectiveness also depends on context: the same control is worth more in a jurisdiction with heavy regulatory exposure, in a sector under active attack, or in an estate with a poor baseline.

The escalation warning at the end is a specific and frequently-observed phenomenon: initial access is rarely the final objective. One compromised workstation becomes domain administrator through credential reuse; one supplier account becomes access to a core network; one exposed administrative interface becomes mass data export. This is why the same asset appears in several domains, why lateral movement matters, and why 'a breach in one domain must not automatically become a breach in another' recurs as a design requirement in the models discussed later.

Practical security

Key concept: Realistic Prevention

- Consider the impact of an attack
  - Under the light of CIA and other potential impact areas (e.g., brand or legal)
- Consider the cost and recover time
  - Data, monetary cost, reputation, market access
- Characterize attackers
  - Define controls specific for those attackers
  - There will always exist more resourceful attackers
- Consider that the system **will be compromised**
  - Have recovery plans assuming that everything else failed

Impact assessment asks what an incident would actually cost, in the light of the three information-security properties and beyond them. Data loss has regulatory consequences under data-protection law, including notification duties and possible fines. Confidentiality loss can destroy competitive advantage or client trust. Integrity loss shows up as incorrect invoices, incorrect medical records, or recalled products. Availability loss stops production, and its cost is usually measured per hour. Brand and legal consequences are frequently larger than the direct technical cost, and they are the hardest to quantify.

Recovery time belongs in the same assessment, because the same event has wildly different costs depending on how quickly service returns. Recovery objectives set in advance, per system, drive architecture decisions and their costs. Restoring from tape, from a cloud snapshot, or from a hot standby are different orders of magnitude, and the difference is only discovered during a real incident if it was never tested.

Characterising attackers is what turns a general list of controls into a proportionate design. An untargeted opportunist using automated scanning tools is stopped by patching, credential hygiene and basic filtering. A criminal group pursuing one specific company uses targeted phishing, purchased or stolen credentials, and patience. A well-resourced actor can exploit unknown software flaws and go after suppliers and administrators directly. Since better-resourced attackers always exist, the realistic goal is not invulnerability but cost: make the effort exceed the likely payoff, and make detection fast enough that an intrusion is interrupted before its objective.

The last bullet is the most important attitude shift in the session. Planning on the assumption that preventive controls will eventually fail changes the design: monitoring and logging become first-class requirements, segmentation limits lateral movement, privileged access is time-boxed and recorded, backups and restore capability are tested, communication and legal steps are prepared, and roles are named in advance. It is a substantially cheaper response to breach than discovering during a breach that none of those exist, and it is the direct conceptual route to the zero-trust model later in this material.

Security in computing systems

Complex problems

- Computers can do much damage in short time frames
  - Computers manage huge amounts of information
  - Process and communicate with very high speed
- The number of weaknesses **is always growing**
  - Due to the increased complexity
  - Due to every reducing time-to-market, or cost

The speed and scale of computing changes the consequences of a fault. A single misconfigured storage permission can expose tens of millions of records in an instant; a bug in a build pipeline can propagate malicious code to every downstream user within minutes; one bad configuration change can take down a global service. In the physical world, a comparable breach exposes what is present in one place; in a networked one, the same mistake is replicated as widely as the system is.

Complexity is the underlying cause. Modern systems are assembled from millions of lines of code across operating systems, libraries, containers, cloud services, firmware and hardware, with interactions no individual fully understands. Every added feature, integration, and convenience adds a code path and a trust assumption, and the number of reachable, unanticipated states grows with the product of components rather than with their sum. Security work at this level is largely about reducing effective complexity: fewer components, fewer trust relationships, fewer privileges, fewer paths, and clearer boundaries.

Commercial pressure makes it worse rather than better. Time-to-market and cost competition systematically reduce the time available for design review, code review, threat modelling, testing and hardening, and those activities are the first to be trimmed when a deadline slips. Features are also often designed to maximise convenience, meaning sharing, defaults that open rather than close, and integrations that grant broad access, all of which are security costs paid later and invisibly.

A useful discussion is whether growing weaknesses are inevitable. The evidence suggests some progress is possible, since memory-safe languages remove whole vulnerability families, managed cloud services reduce the number of components a company must secure, and standard hardening baselines remove configuration errors at scale. But the total surface grows with functionality, so the realistic conclusion is the one from the previous slides: security work is continuous risk management, not a one-off achievement, and it must be prioritised rather than exhaustive.

Security in computing systems

Complex problems

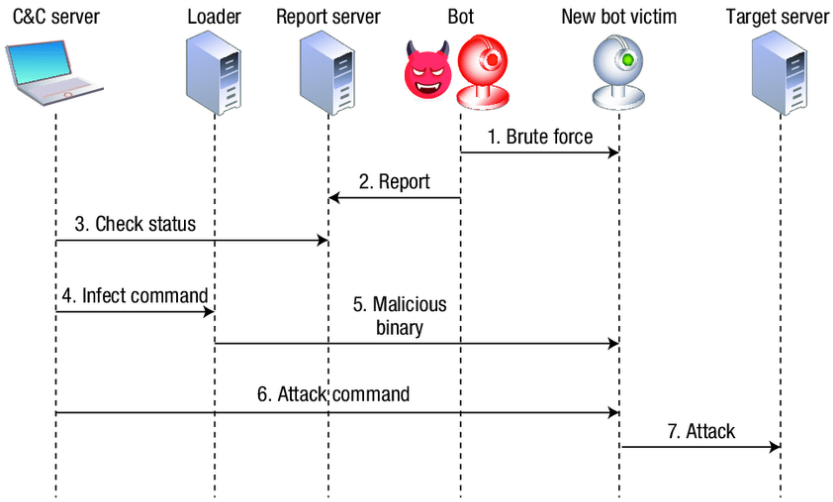
- Networks allow **novel attack** mechanisms
  - "Anonymous" attacks from any place in the planet
  - Fast spread across geographical boundaries
  - Exploitation of insecure hosts and applications
- Attackers can build **complex attack chains**
  - First exploration
  - Lateral movement
  - Exfiltration
  - Check: https://attack.mitre.org/matrices/enterprise/

Networks changed the geometry of attack. Distance and jurisdiction stop protecting anything: an attacker can probe from another continent, route through compromised machines in several countries, and target a machine in a country where they have no presence. Response is slowed by that same geometry, since legal cooperation across borders is slow and attribution is hard, and since a criminal can be operating from a country that will not act.

Speed also changes containment. Malware and exploits spread across the internet in hours, and a vulnerability disclosed publicly typically sees scanning traffic within a day or less, with active exploitation following soon after. That compresses the patching window to a matter of hours for internet-facing systems, and it is why exploit-in-the-wild, rather than the age of a flaw, is what drives emergency patch cycles.

The attack chain described here, initial access, exploration, lateral movement, exfiltration, is the standard shape of a targeted intrusion, and it is documented publicly in a widely used knowledge base of attacker tactics and techniques, MITRE ATT&CK. Its practical value is that it gives both sides a shared vocabulary: defenders map their detections against the techniques to find blind spots, and this mapping exercise is one of the most productive things a security team can do with a mature toolset.

Two of the stages deserve separate attention. Lateral movement is how a small foothold becomes a large compromise, using credential theft from memory, pass-the-hash style reuse, remote administration protocols, trusting relationships between systems, and misconfigurations such as local administrator rights everywhere. Exfiltration is the hardest stage to detect on a well-monitored network because the data must leave, and attackers blend it with legitimate traffic, for example encrypted channels to cloud services, DNS tunnelling, or slow drip feed over days. Defences against the chain as a whole are least privilege, segmentation, monitoring of authentication anomalies and unusual data volumes, and blocking administrative protocols from ordinary workstations.



Mirai botnet operation and communication
Causes Distributed Denial of Service (DDoS) attacks to a set of services, by constantly propagating to weakly configured IoT Devices. Observe that victims are used to conduct further attacks to other victims

source: Kolias, Constantinos et al. "DDoS in the IoT: Mirai and Other Botnets." Computer 50 (2017): 80-84.

The diagram illustrates a botnet built from internet-connected devices with default or trivial credentials, and it captures the essential escalation: each compromised device scans for the next, so growth is exponential and self-sustaining, and the victims of the first stage become the attackers of the second. That structure is what made this class of malware effective, since no user interaction was required at all, and since the infected population was distributed across residential networks with generous bandwidth.

The scale consequences were immediate and visible. Attacks of unprecedented volume were directed at major online services and at DNS providers, taking down widely used websites not by breaching them but by making their infrastructure unreachable. Because the traffic came from hundreds of thousands of ordinary devices, including many that looked legitimate, filtering was difficult. And because the devices belonged to people and companies who had never heard of the malware, the abuse was invisible at the source.

The root cause is instructive rather than historic. Cheap connected devices shipped with hardcoded credentials, no mechanism to change them, no automatic update path, and administrative interfaces exposed to the public internet. Manufacturers had little incentive to spend on security, no revenue stream to fund long support, and often no way to reach devices already sold. Some of this has improved through regulation and certification schemes that require unique passwords, update mechanisms, and vulnerability disclosure policies, and the European Cyber Resilience Act is one of the more significant moves in that direction, but a very large installed base remains unpatchable.

The source code was released publicly shortly afterwards, which is why the pattern repeated: other groups forked it and added new exploits, and variants continued to appear long after the original author was identified. The general lesson is that malware code is a durable asset once published, that IoT and operational technology must be treated as first-class hosts on the network rather than as appliances, and that exposure management, meaning actively finding what is reachable from outside, is one of the highest-value defensive activities there is.

Security in computing systems

Complex problems

- Users are mostly **unaware** of the risks
  - They do not know the problems,
  - ... the impact
  - ... the good practices
  - ... nor the solutions
- Users are **careless**
  - Because they take risks
  - Do not care (do not have/identify any responsibility)
  - Do not estimate the risk correctly

Users are described unsympathetically here, and it is worth reframing, because the framing changes which interventions work. Most users are not careless in the abstract; they are people optimising their own task under competing pressures, for whom security is an unfamiliar domain with invisible consequences and visible inconvenience. When a secure process is slower and its benefit is invisible while an insecure shortcut is faster and its cost appears much later to someone else, predictable behaviour follows.

The awareness gap has four parts and each needs a different intervention. People do not know the problems, so awareness must be concrete and role-specific rather than generic. They do not know the impact, so examples of consequences, real and recent, work better than abstract warnings. They do not know good practices, so the organisation must provide them explicitly and make the secure option the default. They do not know the solutions, so tools must be provided and supported rather than assumed; asking people to be careful without giving them a safe way to share files, for instance, produces exactly the unsafe workaround expected.

Responsibility is a genuine factor, and it is organisational as much as individual. If nobody can say who is accountable for a particular dataset, individuals treat protecting it as somebody else's job. If an incident produced no consequence in the past, vigilance looks unnecessary. If an employee is rewarded only for throughput and never for caution, security signals are weak. Well-run organisations make responsibility concrete, for example by naming data owners, by tying incidents to real feedback rather than blame, and by showing that management follows the same rules.

Misestimating risk is a documented cognitive pattern: people correctly acknowledge that a threat exists in general while believing their own likelihood is lower, and they judge risk by vividness and recent personal experience rather than statistics. Defences against this are not lectures but repeated, low-friction practice, such as short simulated exercises, immediate feedback, and prompts at the moment of decision, for example a warning when sending a large attachment outside the organisation or when a file is about to be shared publicly. The measurable goal is a reporting culture: the useful metric is not how many people avoid clicking, but how many report.

Main sources of issues

- Hostile applications or bugs in applications
  - Rootkits: Insert elements in the operating system
  - Worms: Software programs controlled by an attacker
  - Virus: Pieces of code that infect other files (e.g., macros)

- Users
  - Ignorant, careless or reckless
  - Use insecure alternatives instead of secure ones
  - Trust on security tools to solve all problems
  - Search and download illegal stuff
  - Hostile

The three families are distinguished by behaviour rather than by name, and the distinction matters because it drives how each is detected. A virus is code that attaches itself to other objects, executable files, documents with macros, boot sectors, scripts, and spreads through use of those objects. A worm is self-propagating software that finds and exploits hosts or contacts by itself, requiring no user action, and spreads through the network. A trojan-style program looks useful and carries hidden functionality. A rootkit is not a propagation mechanism at all but a hiding mechanism: it modifies or subverts the operating system so that the attacker's presence is not visible to the system's own tools.

Detection follows from that classification. Worms reveal themselves through network behaviour, scanning patterns, unusual authentication attempts and traffic volumes, often before their payload is analysed. Viruses are found by inspecting files and their changes, by signature and behavioural analysis, and by looking at persistence points such as startup entries, scheduled tasks, browser extensions, and macros set to run automatically. Rootkits are the hardest, because the tools used to look for them are the tools being subverted; reliable inspection therefore uses an external viewpoint, for example booting from trusted media, comparing files against known-good hashes, using hardware-backed measurements such as trusted boot and measured boot, or inspecting a hypervisor layer that the malware cannot control.

Malware today is mostly commercial and mostly delivered through legitimate-looking channels: phishing attachments and links, malicious or compromised advertisements, fake installers and fake updates, repacked software from unofficial download sites, and abused remote administration tools that are legitimate products signed by their vendors. Because the industry knows this, defensive focus has moved from signature-matching payloads to blocking delivery paths and behaviours: mail filtering, link rewriting, download reputation, application control, restricting scripting engines, and monitoring for suspicious process behaviour such as a document spawning a shell.

The user-related entries deserve careful handling in class. Using an insecure alternative to a secure one, for example a personal file-sharing service or consumer messaging for work data, is usually a rational response to friction and is best addressed by making the sanctioned tool genuinely better. Trusting security tools to solve everything produces dangerous overconfidence, and tools that are unmanaged or unmonitored can be turned off by malware or left in permissive mode. Downloading illegal material brings genuine risk, since cracked software and key generators are a major malware vector and are frequently bundled with credential stealers. And 'hostile' describes the deliberate insider, whose harm is intentional, is not fixable by training, and requires least privilege, monitoring, separation of duties and, when suspicion arises, a properly handled and legally careful investigation.

Main sources of issues

- Defective administration
  - Default configuration is seldom the most secure
  - Security restriction vs flexible operation
  - Exceptions to individuals
- Communication over uncontrolled/unknown network links
  - Public hotspots, campus networks, hostile governments

Default configuration is designed to make products work out of the box, which means it is designed for connectivity and convenience rather than least privilege. Typical consequences include administrative interfaces reachable from the network, default or shared credentials, verbose services enabled, unencrypted protocols still accepted, sample pages and debug endpoints exposed, telemetry and remote-support features enabled, and logging either off or verbose enough to disclose secrets. That is why hardening benchmarks exist as a genre: standardised lists of settings to change on a specific product or platform before use, and why configuration drift, meaning settings quietly returning to defaults after updates or new deployments, is itself monitored.

Restriction versus flexibility is the recurring tension in administration, and it is resolved through exceptions with a lifetime. Every exception, an opened port, a shared account, a user with local administrator rights, a disabled antivirus agent, is a documented deviation with an owner, a justification, an expiry date and a review. Programmes that do not track exceptions tend to accumulate thousands of them, and audits read that list directly as the organisation's real security posture, since exceptions are usually where attackers look first.

Uncontrolled network links deserve updating in light of current practice. The principle, do not trust a network you do not control, has not changed, but the specific risks have. Opportunistic eavesdropping is now much less effective because most major services use transport encryption by default, and certificate validation makes simple interception visible. In its place sit rogue access points with familiar names, captive portals designed to harvest credentials, forced-downgrade attempts, and malicious or compromised hotspot infrastructure. The practical defences are VPN or zero-trust access over untrusted networks, certificate-based authentication rather than password logins, and not treating network location as proof of identity.

There is a broader administrative lesson in this slide: most successful intrusions use features working as designed. Remote administration tools, scheduled tasks, legitimate scripting interpreters, cloud storage, and valid credentials do not look like malware because they are not. That is why administration quality, patch currency, configuration baselines, exception hygiene, credential lifecycle and monitoring coverage are treated as security controls in their own right, and why 'defective administration' is such a common root cause in post-incident reports.

Perimeter Defense Model

Minimal defense, frequently not sufficient. The most common.

The model shown here is the historically dominant one, and its logic is simple: define a boundary between a trusted inside and an untrusted outside, and concentrate protection at that boundary. It is cheap, easy to explain, easy to purchase, and it works surprisingly well against the least sophisticated threats, which is why it remains the most common arrangement and why it is described here as minimal defence rather than as a mistake.

Its mechanism is the firewall, a gateway that filters traffic between domains according to rules. Practical firewall behaviour includes stateful inspection, meaning responses to established connections are allowed while unsolicited inbound traffic is dropped, network address translation, which incidentally hides internal addressing, and application-layer filtering. A single device typically also performs routing, virtual private network termination, and sometimes content filtering, which concentrates a great deal of trust and a great deal of risk in one place.

The model was built for a world in which users worked inside a building, servers were inside a data centre, and the internet was a single link. Cloud services, remote working, mobile devices, and interconnection with partners and suppliers broke the assumption that there is one meaningful boundary. When applications are hosted externally and staff work from anywhere, the boundary is no longer a place but a set of identities, and the diagram on this slide starts to look like a description of something that no longer exists.

There is also a subtle cost to the single-boundary habit: because the boundary is the only defended point, everything else tends to be configured on the assumption that it is unnecessary. Network access becomes a strong implicit credential, so being plugged into a port confers trust, and administrative interfaces are left exposed internally because 'only insiders can reach them'. Those two habits are responsible for a great deal of damage during lateral movement, and they persist long after the perimeter itself has stopped being the primary control.

Perimeter Defense Model

- Protection against external attackers
  - Internet
  - Foreign users
  - Other organizations
- Assumes that internal users are trusted and share the same policies
  - Friends, family, collaborators
- Used in domestic scenarios or small offices
- Limitations
  - Too simple
  - Doesn’t protect against internal attackers
    - Previously trusted users
    - Attackers that acquired internal access

The model's assumptions should be read as a list of claims to be tested. External attackers are the primary threat, internal users are trustworthy, and everyone inside shares the same policies. Each of those was more plausible in a small office than in a large modern organisation, and none is a fact of nature. The model is genuinely appropriate for the domestic and small-office case named here, where the estate is small, homogeneous, and operated by people who know each other.

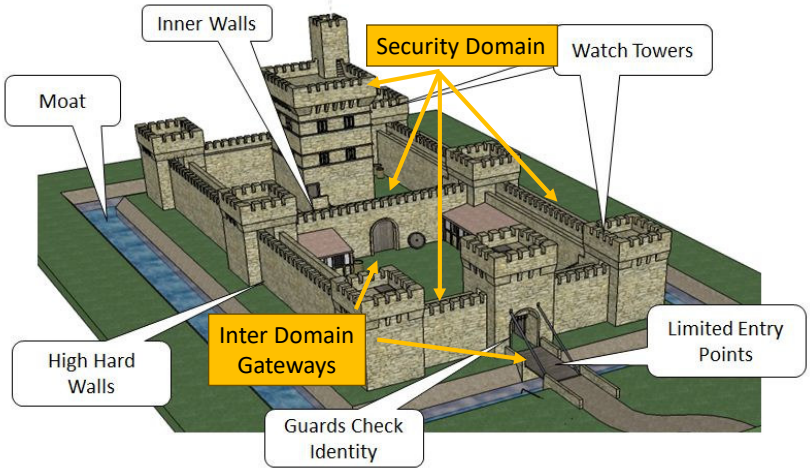
Its principal weakness is often called a hard shell and soft interior. Once an attacker is inside, commonly through a phishing email, a malicious attachment, a compromised supplier connection, or a laptop used both at home and at work, there is often little standing between that position and the valuable assets, because internal traffic was never designed to be restricted or inspected. Credential theft then scales the intrusion quickly, since a single stolen administrative credential can confer access across the entire interior.

Insider risk is the second weakness, and it appears in all three forms: the malicious insider who was always trusted, the compromised insider whose credentials or machine were taken, and the careless one who connects an infected personal device to the internal network. A model that assumes internal users are trusted has no answer to any of them, and neither does it answer the ordinary case of a mistake, such as an administrator moving a server into the wrong network segment and thereby granting it broad access.

The limitations listed here are exactly what motivates the next two models. If one boundary is insufficient, add many internal boundaries and put controls at each, which is defence in depth. If the very idea of inherent interior trust is unsound, stop granting it and evaluate every request on its own evidence, which is zero trust. It is worth being able to state that progression in one sentence, since it is the narrative thread of the rest of this session.

Defense in Depth Model

Layered approach with multiple domains (better)



The term comes from military doctrine, where the idea was to trade space for time, absorbing and slowing an advancing force rather than staking everything on a single defensive line. Applied to systems, it means deploying several independent layers so that defeating one does not yield the objective, and each layer buys time and produces evidence. The concept predates modern security practice, and its vocabulary, including the phrase 'defence in depth' and related ideas such as onion-style layered protection, is used across many security frameworks.

Layers can be of different kinds, and a good design mixes them. Physical layers are walls, doors, cabinets and cameras. Network layers are segmentation, firewalls between zones, and filtering at hosts. Identity layers are authentication, single sign-on, and privileged access management. Host layers are hardening, endpoint protection, and application control. Application layers are input validation, secure coding, and logging. Data layers are encryption, integrity protection, and backups. Process layers are review, audit, separation of duties and training. Mixing kinds matters because an attacker strong against one kind is not automatically strong against another.

The value of a layered design comes from independence, and independence has to be designed deliberately. If the same administrator, the same directory service, the same vendor console, and the same network path protect all the layers, then compromising that one thing compromises all of them, and the design is one control with several interfaces. This is why security boundaries are ideally owned by different teams, use different credentials, and do not share infrastructure with what they protect.

A useful way to evaluate any architecture, in class or at work, is to ask what an attacker who has already defeated a given layer must defeat next, and how long that would take and how visible it would be. If the honest answer is 'nothing, or something instantaneous and silent', then that is a gap in the layering rather than a property of the attacker.

Defense in Depth Model

- Protection against internal and external attackers
  - From the Internet
  - Users
  - Other organizations
- Assumes well-defined domains across the organization
  - Walls, doors, authentication, security personnel, ciphers, secure networks
- Limitations
  - Needs coordination between the different controls
    - May end with overlapping controls, but also with holes in the security perimeters
  - Cost
  - Requires training, changes to processes and frequent audits

Defence in depth changes the assumption, from 'the boundary holds' to 'the boundary will be crossed eventually, and crossing it must not be enough'. Both internal and external adversaries are therefore in scope, and the interior acquires structure: zones, domains, gateways between them, authentication at several points, monitoring inside as well as at the edge. The controls listed, walls, doors, guards, passwords, ciphers and secure networks, are all boundary controls, but now they are replicated at each internal boundary rather than only at one.

The design requires well-defined domains, and that is a substantial organisational commitment rather than a purely technical one. Someone must decide what zones exist, what belongs in each, what may cross, and who owns each boundary. In practice the zones emerge from departmental boundaries, and departments change, merge, and build their own connections, so the architecture drifts towards a mesh in which the intended structure is no longer documented or understood. Audits of large organisations frequently find firewall rules that no one can explain being kept because no one dares remove them.

The coordination limitation is real and is usually organisational in origin. Network controls are run by the network team, endpoint controls by the desktop or platform team, identity controls by the directory team, application controls by developers, and cloud controls by yet another group, each with their own tooling, change process and on-call rota. Gaps open precisely between them: a host exempted from endpoint protection by the developers, an inbound firewall rule opened for a project that was never closed, a cloud account outside the standard baseline. Overlap is the other failure mode, where several controls each assume the other is handling something, and legitimate traffic breaks.

Cost and maintenance are the honest drawbacks, and they should be stated plainly. More boundaries mean more devices to buy, configure, patch and monitor, more rules to review, more change requests, more training, and more auditing to keep the structure truthful. It also increases the chance that someone works around a boundary rather than through it. The trade-off is not 'more security for free', it is buying measurable reduction in blast radius with ongoing operational effort, which is why the amount of layering should be proportionate to the value of what is inside.

Zero Trust Model

- Defense model without specific perimeters
  - There is no inherent trust in entities just because they are internal
    - Actually, there may be no notion of internal and external
  - Requires detailed knowledge, controls and observability between all entities
- Model recommended for new systems
  - Traditional systems should migrate to it
  - Implies the design of systems/services specific for this model
  - Legacy systems will need additional protection layers
    - Firewalls, filters, adapters, plugins

Zero trust abandons the premise shared by the previous two models, that location confers trust. In a zero-trust architecture, a request is authorised based on evidence about the identity, the device, the service and the resource, evaluated for every request, and not on the fact that the request came from a network address considered internal. The word 'trust' in the name is slightly misleading, since the model is not distrust of people but a refusal to grant standing, network-based trust.

Practically, this means control moves from the network to identity and policy. Access decisions combine who is requesting, from which device with what security state, to which resource, with what sensitivity, under what conditions, and consult a policy engine. Micro-segmentation limits which hosts may talk to which, service-to-service calls authenticate with strong credentials rather than by IP address, and privileged access becomes just-in-time and recorded rather than permanent.

The requirements listed, detailed knowledge, controls and observability between all entities, are the reason adoption is hard. A policy engine needs to know what services exist, who uses them, and what they legitimately do. Most organisations cannot answer that accurately today, which is why implementations typically start by observing traffic to learn normal behaviour, and only later enforce it. This learn-then-enforce sequence is where many projects stall, because observation phase lasts longer than expected and enforcement gets postponed indefinitely.

The migration advice deserves emphasis, since it is where most real programmes differ from the marketing. New systems are designed for the model, with identity-aware services and no assumption of interior trust. Existing systems rarely support it directly and are wrapped instead: retained perimeter controls, gateways and proxies in front of legacy applications, adapter layers, virtual private network access limited to specific resources, and reverse proxies that add authentication where the application cannot. Legacy is not usually an argument against the direction, it is an argument for a long, staged programme measured in years, and for keeping defence in depth in place around the parts that cannot change.

Zero Trust Model - Principles (NCSC)

1. Know your architecture
   - Users, devices, services and data
2. Know your identities
   - Users, devices, services and data
3. Assess user behaviour, service and device health
   - Accept but verify if everything is ok
4. Use policies to authorize requests
   - Define what is legal and illegal

The first principle, know your architecture, sounds trivial and is not. It means an accurate inventory of users, devices, services and data, and of how they talk to each other, kept current. Asset discovery is a security product category precisely because most large organisations do not have this, and because undocumented systems, forgotten administration interfaces, and shadow IT, meaning services adopted without institutional approval, routinely turn up during incidents and audits. Without the inventory, every subsequent principle is guesswork.

The second principle, know your identities, extends identity beyond human accounts to every subject that can act: service accounts, machine identities, certificates, API keys, devices and the services themselves. Non-human identities typically outnumber human ones by a large factor in modern environments, they are frequently shared, they often never expire, and their secrets live in code repositories, configuration files and pipeline variables. Key and secret lifecycle management is therefore a first-order concern in a zero-trust design rather than a housekeeping detail.

Assessing behaviour and health means continuously evaluating context rather than checking once at login. Is the device managed, patched, disk-encrypted, and free of obvious compromise? Is the login from an unusual location, at an unusual hour, or at a rate inconsistent with human behaviour? Is the request for data disproportionate to the role? Access decisions become conditional and revocable, which is why single sign-on sessions, token lifetimes, and step-up authentication matter operationally in this model.

'Accept but verify' is worth unpacking as a deliberate rewrite of the older phrase 'trust but verify'. In the older framing, some baseline trust is granted and then checked occasionally; in zero trust there is no baseline grant, and verification is continuous, so an access decision can be withdrawn mid-session when evidence changes. That has concrete architectural consequences: short-lived credentials rather than long-lived secrets, session re-evaluation, and enforcement points that can terminate access immediately rather than waiting for a logout or a network change.

Zero Trust Model - Principles (NCSC)

5. Authenticate and Authorize everywhere
   - No open APIs, or IP address-based access

6. Focus your monitoring on users, devices and services
   - Develop the capability to observe interactions

7. Don’t trust any network, including your own
   - Internal attackers should not have more rights than external attackers

8. Choose services designed for Zero Trust
   - Legacy services to be avoided, but can be integrated

No open APIs and no IP-address-based access is a direct rejection of two very common legacy practices. IP allow-listing grants access based on an address, which proves nothing about identity since addresses are shared through network address translation, reassigned, spoofable in some contexts, and routinely reachable from inside a network; and it silently breaks when addresses change, which is common in cloud environments. Replacing it with identity-based authorisation, strong device-bound credentials and signed service-to-service authentication is one of the more concrete differences between a zero-trust design and an older one.

Authorising everywhere means there is no 'internal' fast path and no management interface reachable merely by being on the right network. Practical consequences include removing default-deny exceptions, requiring authentication for administrative planes, and eliminating shared service accounts in favour of per-service identities with narrowly scoped permissions, which also makes the resulting logs interpretable.

Focusing monitoring on users, devices and services reflects that in this model the interesting events are interactions between subjects and resources, not traffic between subnets. That means collecting authentication and authorisation decisions, including denials, privilege use, configuration changes, and service-to-service call patterns, and having the capability to query them at investigation speed. Building that capability is often the largest single investment, and it is also what makes the model defensible: without observability, a zero-trust policy is enforced but not understood.

The last two bullets carry the practical wisdom. Not trusting any network, including your own, is the sentence that most sharply separates this model from the previous two, and its corollary is that encryption and authentication are required for internal traffic as well as external traffic, which legacy estates frequently resist because of performance and complexity. Choosing services designed for the model matters more than it sounds: products that require broad network access, that authenticate by address, or that assume a flat trusted network will need wrapping and compensating controls. Legacy can be integrated, but the cost should be counted explicitly, and the decision to keep it should have a review date attached.

In practice?

- Cybersecurity is limited by economics, operations and logistics
  - All entities have limited resources
    - Even attackers!
  - Security is a business continuity activity, it cannot prevent business
- Cybersecurity deals with building and applying a strategy
  - under an operational and legal context
  - preventing issues that may never happen

The closing argument is that security is bounded by resources, and that attackers have resource limits too. Defence must be paid for out of the same budget as everything else, and attacker effort, tooling and time are finite, so a rational defensive strategy raises the attacker's cost above the value of the target and reduces the chance that any given attempt succeeds within their patience and budget. This reframes security from an absolute goal to a continuous economic argument, which is how practitioners actually make decisions.

Security serves business continuity and cannot be allowed to prevent business, which is a genuinely difficult line to hold. Controls that stop the business from operating are eventually bypassed, waived or removed, so a control whose cost in friction exceeds its expected benefit tends to be destroyed by the organisation rather than by the attacker. Good design makes the secure path the ordinary path, and treats each exception as information about where the design failed rather than as evidence of a difficult user.

Security work is also unusual in that success is invisible: prevented incidents do not happen, and there is no natural event to point at. This produces a real management problem, since investment competes with projects that have demonstrable output. The usual response is to measure proxies rather than outcomes: mean time to detect and to respond, coverage of monitoring and of patching within required times, percentage of systems meeting hardening baselines, results of exercises and tests, reduction in the number of outstanding exceptions, and the ratio of internally-reported to externally-reported incidents. None is perfect, but a programme without any such indicators cannot argue for its own budget.

The final framing places all of this inside an operational and legal context: controls are chosen under constraints of legacy systems, staffing, contract obligations, labour and privacy law, data-protection duties, cross-border rules, and sector regulation, and they are chosen to prevent issues that may never occur. That is what makes the discipline closer to risk management than to engineering in the classic sense, and it is a reasonable summary of what a security professional actually spends time doing: making explicit, defensible choices about which risks to reduce, how far, at what cost, and how to know whether it worked.