

Securing Devices

AP Cybersecurity

Device Vulnerabilities and Attacks

A **device** is any computer - a server, a personal laptop, a smartphone, or an **embedded computer** built into a machine. Everyday devices with embedded computers are called **Internet of Things (IoT)** devices, and they run everything from water pumps to washing machines.

■ The four classes of device, and why the class matters

Class	What it is	Security consequence
servers	shared machines running services for many users	the highest-value target; one compromise reaches everyone
personal computers	desktops and laptops	general purpose, so they run anything the user installs
handheld computers (also called mobile computers or information appliances)	smaller than a PC and running on battery power – smartphones, tablets, smart watches and other wearable technology	easily lost or stolen, and often carried across untrusted networks
embedded computers	a computer that is part of a machine – a car's engine controller, a thermostat, a medical pump	has a specialised instruction set for interfacing with its components, and tends to be slower, cheaper and to have minimal storage , so security features are often left out and updates are rare

That last row is the reason embedded and IoT devices appear so often in attack scenarios: the constraints that make them cheap are the same constraints that make them hard to defend.

The main threat to a device is **malware** - malicious software. Learn the types:

- **Virus** - attaches to a host file or program and runs when that host executes; opening an infected file is one possible trigger.
- **Worm** - spreads by itself, with no human action.

- **Trojan (RAT)** - hides inside software that looks safe; a **remote access trojan** gives the adversary remote control.
- **Ransomware** - encrypts your files and demands payment for the key.
- **Spyware** - secretly tracks what you do.
- **Keylogger** - records every keystroke to steal passwords.
- **Logic bomb** - triggers only when a condition is met (a date, a version).
- **Rootkit** - deeply hides in the operating system and can even make itself invisible.

Most malware is a file, but **fileless malware** is different: it lives only in **RAM** and abuses legitimate programs already on the device, leaving no file for a scanner to find.

Adversaries exploit **unpatched software**, weak passwords, unprotected **BIOS/UEFI** startup settings, and open ports. We rate device risk by the value and criticality of the device - a hospital's unpatched email server is **high** risk, while an employee's laptop with one unused open port is **low**.

Authentication



A fingerprint scanner: biometric authentication checks something you ARE, which is much harder for an attacker to steal or guess than a password

To store passwords safely, systems use a **cryptographic hash function**

- a one-way maths algorithm that turns any input into a fixed-length string called a **hash** (or digest). Hashes have three vital properties: they are **collision resistant** (hard to find two inputs with the same output), have **pre-image resistance** (recovering an input from a hash is computationally infeasible in general, although weak passwords can be guessed), and are **repeatable** (the same input always gives the same hash).

Try it – read real hash results. These inputs use UTF-8 with no final newline. Only the first 16 of each full SHA-256 hash’s 64 hex characters are displayed.

Input	Hash prefix	Full hex length
cat	77af778b51abd4a3	64
cat again	77af778b51abd4a3	64
cats	d936608baaacc6b7	64

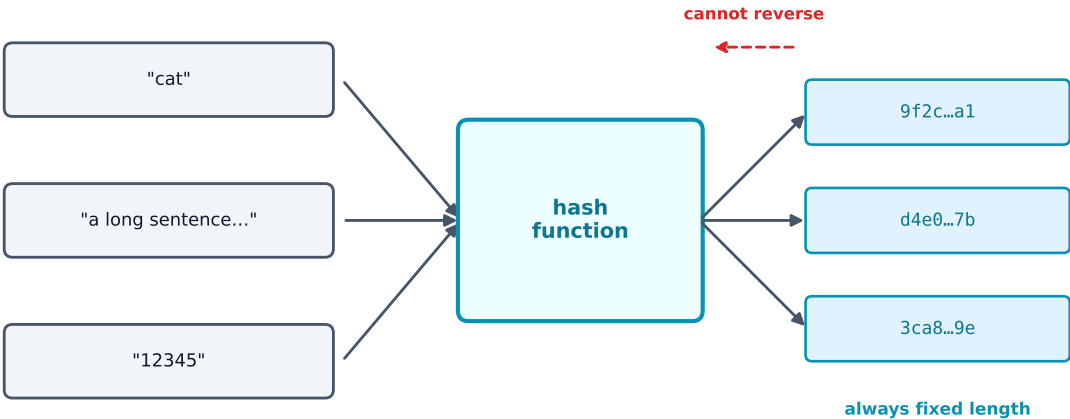
Which rows illustrate repeatability? What stays fixed when input length changes? Do these results prove that collisions are impossible?

Worked reasoning. The two identical `cat` inputs give the same full hash. Both `cat` and `cats` have outputs of 64 hex characters. Each hex character represents 4 bits:

$$b = 4h = 4 \times 64 \text{ bits} = 256 \text{ bits}$$

The changed output for `cats` illustrates sensitivity to an input change; three examples do not prove collision resistance. Matching prefixes alone do not establish that full hashes match. The full outputs in this worked example were checked.

Why collisions exist. An n -bit output has $N = 2^n$ possible values. A toy 4-bit hash has $N = 2^n = 2^4 = 16$ outputs. With 17 different inputs, at least two must share an output. This toy is not secure. Real cryptographic hashes have much larger output spaces, but collisions still exist; the goal is to make finding them computationally infeasible.



AP Cybersecurity · U4 · a one-way hash function -> fixed-length digest

Schematic: the digest labels are illustrative, not computed SHA-256 values. There is no direct decryption step; guesses can still be hashed and compared.

Real hash functions have names. The **Secure Hash Algorithm (SHA)** family – SHA-256 and SHA-512 – is today’s standard. Adversaries attack a hash function by trying to **force a collision** (two different inputs with the same hash); once an

efficient collision attack exists, that function is **deprecated** (retired from secure use). **MD5** and **SHA-1** are the classic deprecated examples – never rely on them to protect data today.

A secure service should not store your plaintext password. It stores a salted password **hash**; when you log in, it hashes what you typed and compares. To stop two identical passwords producing identical hashes, a few random bits called **salt** are added before hashing, so identical passwords do not normally produce identical stored hashes. Salt is stored with the hash and does not need to be secret.

Worked example. Two users both choose the password **sunshine**. Without salt, both stored hashes would be identical, so cracking one instantly cracks the other. Give each user a unique random salt - shortened here to **x7** and **q2** for readability - and the service hashes **sunshinex7** and **sunshineq2** instead. The two stored hashes now look completely different, so the adversary must attack each account separately. Salt prevents reuse of one unsalted lookup table across accounts, but does not make a weak password strong. This simple concatenation illustrates the idea; real password storage uses a purpose-built slow password-hashing scheme with a suitably sized unique random salt, not raw SHA-256.

Transfer – one-way does not mean unguessable. A stolen record has salt **x7** and a SHA-256 hash. Its displayed prefix is **f9325402a7a1542a**. An adversary guesses **rainbow** and **sunshine**, appends the salt, hashes the results, and compares them with the record. The **rainbowx7** prefix is **2a0e6ce47f2c701d**; **sunshinex7** begins **f9325402a7a1542a**. Which guess should be checked in full? Was the hash decrypted? Would live account lockout stop these computations?

Worked answer. The full **sunshinex7** hash matches the stored value; the full **rainbowx7** hash does not. This is guessing, hashing and comparing, not decryption. The salt was known, and the weak password was still guessed. Account lockout controls live login attempts; it does not stop computations on a stolen database. The guesses generate no login events at the service, although database theft or later use of the credentials may leave evidence.

Adversaries fight back with **password attacks**. **Online** attacks submit credentials to a live login; **offline** attacks steal the hash database and crack it on their own machine (which bypasses any account-lockout protection). Techniques include:

- **brute force** - an automated tool tests all candidates in a chosen search space. Exhaustive search finds a match if the space includes the password and every candidate can be tested, but the cost may be impractical. For length L with A choices per position, the candidate count is $N = A^L$. Real attacks often try likely choices first.
- a **dictionary attack** - the tool tries a list of common words and known passwords first, because most people pick guessable ones.
- **password spraying** - one common password against many accounts (this dodges lockout, which counts failures per account).

- **credential stuffing** - reusing stolen or default credentials, exploiting that people reuse passwords across sites.
- a **rainbow table** - a precomputed table of passwords and their hashes, sorted by hash, so a captured hash can be looked up instead of recomputed.

■ Password policy settings

An administrator hardens accounts by **configuring login settings** - and the exam expects you to name them and say what each defends against:

Setting	What it does	The attack it slows
complexity	require a character from each set (upper, lower, digit, special)	brute force / dictionary
minimum length	require N characters - length matters more than anything	brute force (grows exponentially)
maximum age	sets a time limit after which a password must change; a policy mechanism, not a universal recommendation	limits how long a stolen password is useful
password history	store the last 5-10 hashes, block reuse	stops recycling an old (possibly leaked) password
lockout	lock the account after 3-5 wrong tries	brute force / online guessing

Distinguish a setting from a recommendation. NIST SP 800-63B-4 says not to require periodic password changes or character-class composition rules. Change a password when compromise is suspected or known, and block common/compromised choices. Routine expiry can encourage predictable patterns like **PasswordFall12028**. A **password manager** solves the real problem - it generates and stores a long, unique password per site, so none is ever reused or guessable.

Authentication factors prove who you are, and fall into categories: something you **know** (a password), something you **have** (a token or phone), something you **are** (a **biometric** like a fingerprint or retina scan), and somewhere you **are** (a location factor). Using two or more is **multifactor authentication (MFA)** - far stronger than a password alone.



A hardware security key proves who you are with something you physically hold – a strong second factor

■ Removable media, and the autorun problem

An adversary can load malware onto an **external drive** – a USB stick, a portable disc – and leave it where someone will pick it up. If **autorun** is enabled, the device runs a program from that drive **the moment it is inserted**, with no click required, so the malware executes before the user has decided to trust anything.

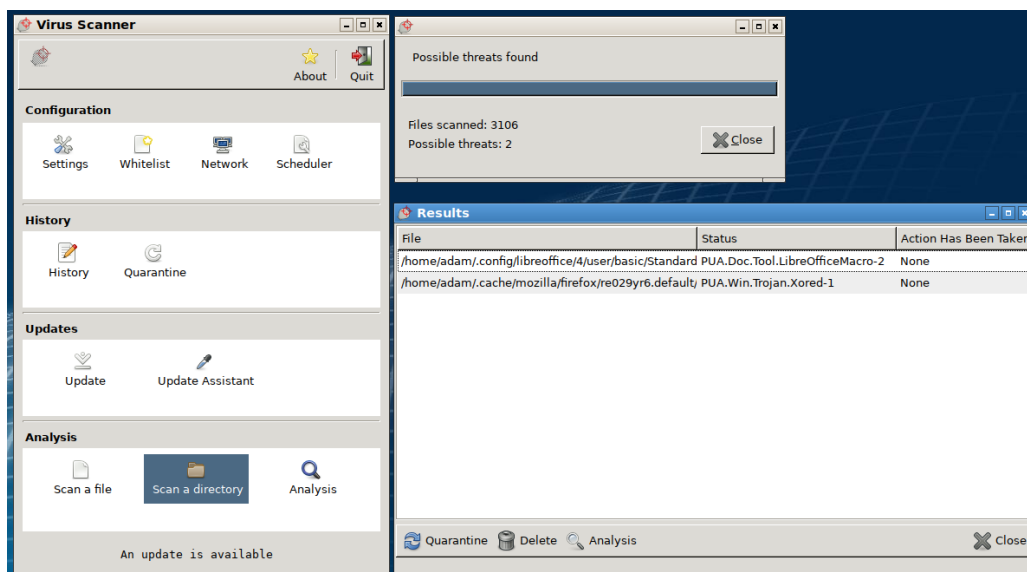
Two controls answer this, and the exam wants both named:

- **Disable autorun**, so inserting a drive never runs anything by itself.
- **Prohibit users from connecting external drives or media** at all – enforced by policy and by a technical control that blocks the USB ports – which is why so many secure environments physically or logically disable them.

Protecting Devices

Managerial controls set the rules: an **acceptable use policy** lists what users may and may not do, a **password policy** sets length and reuse rules, and a **software installation policy** controls what can be installed.

Technical controls do the work. **Anti-malware software** keeps a database of malware **signatures** and quarantines any file that matches. Keeping the operating system and applications **updated** - installing each **patch** - closes known holes before adversaries can use them. A **host-based firewall** controls traffic in and out of one single device, blocking ports and services it does not need.



Anti-malware software scans files against a signature database and quarantines any matches – this scan has flagged two threats

Detecting Attacks on Devices

Devices log logins, file changes, and processes, and these logs reveal an **indicator of compromise (IoC)** – evidence that an adversary got in. **Host-based** IoCs show up as unexpected processes or changed settings; **file-based** IoCs are files whose hash matches known malware; **behaviour-based** IoCs are things like many failed logins or unusual login times.

Choosing a detection method means weighing **performance** (signature-based is lighter, better for weak devices), **cost** (an **endpoint detection and response (EDR)** service is powerful but expensive), and how sensitive the device is. Reading **authentication logs** exposes password attacks: many wrong passwords for one user signals a guessing attack; many users failing in quick succession is an indicator of **password spraying**; the AP course document also treats rapid default-credential attempts as an indicator of **credential stuffing**. These patterns need context; one shared IP address does not by itself prove an attack. Offline guesses create no live login events because they run on the adversary's copy; the theft or later use of the credentials may still be detected.

Speed is itself a security factor. **Signature-based** detection compares what it sees against a list of known-bad patterns, so it is **faster** than **anomaly-based** detection, which must first learn what normal looks like and then measure every event against that model. Anomaly-based detection catches attacks that have no signature yet, but it costs far more processing power – and on a device that lacks it, the effect compounds: the detection runs slowly, the device degrades, and the method ends up not being implemented effectively at all.

Exam tips

- Know each **malware type by its defining trait**: a worm self-spreads, a virus attaches to a host and runs when the host executes, ransomware encrypts for money, a RAT gives remote control, a rootkit hides.
- A hash is **one-way and fixed-length**; a unique **salt** changes the hashed input even when passwords match. A secure service should store a **salted password hash**, using a purpose-built slow scheme rather than raw SHA-256.
- Name real algorithms: **SHA-256/SHA-512** are current; **MD5** and **SHA-1** are **deprecated** because efficient collision attacks exist.
- Match the password attack to its log signature: cite failed-login entries, timing and users. These are indicators, not proof. The AP CED includes default-credential attempts under **stuffing**.
- Sort authentication factors into **know / have / are / where**, and remember **MFA** combines two or more - a fingerprint plus a password, not two passwords.
- **Offline guessing creates no live login events**: the computations happen on the adversary's copy. Theft or later account use may still leave evidence.

Course terminology and implementation boundary. The AP CED 4.2.B.5 includes default-credential attempts under credential stuffing. OWASP's narrower definition focuses on replaying stolen username/password pairs across services. Use the AP definition for AP questions, while identifying the actual evidence and mechanism. The course's named general-purpose hashes illustrate hashing; they are not a recommendation to store passwords with plain SHA-256. See OWASP password-storage guidance and NIST hash properties.