Question: Equifax Multi-Stage Breach (Initial Access → Exfiltration)

Scenario

In 2017, an enterprise deployed an Apache Struts–based web application on Amazon EC2 in a public subnet behind an Elastic Load Balancer (ELB) terminating TLS. DNS was provided by Route 53. A private subnet hosted an RDS PostgreSQL database containing PII. AWS Inspector ran vulnerability scans (misconfigured and producing false negatives), and GuardDuty acted as an IDS but had its certificates expired, disabling alerts. CloudWatch Logs fed into an Elastic-based SIEM, but no micro-segmentation existed between web and DB tiers. Attackers exploited the unpatched CVE-2017-5638, executed code on EC2, pivoted via SSH to the database host, extracted PII, and exfiltrated it over an encrypted C2 channel.

Tactics Used (MITRE ATT&CK)

- **Initial Access (T1190)**: Exploit Public-Facing Application
- **Defense Evasion (T1562.001)**: Disable IDS
- **Discovery (T1087)**: Account Discovery
- **Lateral Movement (T1021.002)**: Remote Services (SSH)
- **Collection (T1005)**: Data from Local System
- **Exfiltration (T1041)**: Exfiltration Over C2

Security Controls (and example tools)

- **Patch Management System** (update orchestration)
- **Automated Patching** (AWS Systems Manager)
- **Web Application Firewall (WAF)** (ELB-level protection)
- **Network Segmentation / Micro-segmentation** (VPC subnets & security groups)
- **Intrusion Detection System (IDS) with valid certificates** (GuardDuty)
- **Vulnerability Scanning** (AWS Inspector correctly configured)

- **Encrypted Data in Transit** (TLS 1.3 east-west)
- **SIEM with CloudWatch Logs** (real-time correlation)
- **Behavioral Analytics (UEBA)** (detects unusual DB access)
- **SOAR Playbooks** (automated containment and remediation)

Question

Which combination of controls, leveraging Defense in Depth (DiD), Adaptive Security Architecture (ASA), and Zero Trust Architecture (ZTA), best prevents the initial Struts exploit, detects lateral movement, restricts unauthorized DB access, and responds to exfiltration?

Options

1.

- **Prevent (DiD)**: Patch Management System + WAF
- **Detect (ASA)**: Vulnerability Scanning + SIEM with CloudWatch Logs
- **Prevent Access (ZTA)**: Network Segmentation + IDS with valid certificates
- **Respond (ASA)**: SOAR Playbooks + Behavioral Analytics

2.

- **Prevent (DiD)**: Automated Patching + WAF
- **Detect (ASA)**: IDS with valid certificates + Behavioral Analytics
- **Prevent Access (ZTA)**: Micro-segmentation + Encrypted Data in Transit
- **Respond (ASA)**: SOAR Playbooks + Certificate Management

3.

- **Prevent (DiD)**: Patch Management System + Automated Patching
- **Detect (ASA)**: Behavioral Analytics + SIEM with CloudWatch Logs
- **Prevent Access (ZTA)**: Micro-segmentation + WAF
- **Respond (ASA)**: SOAR Playbooks + IRT Coordination

4.

- **Prevent (DiD)**: WAF + Vulnerability Scanning
- **Detect (ASA)**: IDS with valid certificates + Automated Patching
- **Prevent Access (ZTA)**: Network Segmentation + Certificate Management
- **Respond (ASA)**: Behavioral Analytics + Encrypted Data Channels

5.

- **Prevent (DiD)**: WAF + Patch Management System
- **Detect (ASA)**: SIEM with CloudWatch Logs + Vulnerability Scanning
- **Prevent Access (ZTA)**: Encrypted Data in Transit + Network Segmentation
- **Respond (ASA)**: Behavioral Analytics + SOAR Playbooks

6.

- **Prevent (DiD)**: Automated Patching + Certificate Management
- **Detect (ASA)**: CloudWatch Logs + Behavioral Analytics
- **Prevent Access (ZTA)**: Micro-segmentation + WAF
- **Respond (ASA)**: IRT Coordination + SOAR Playbooks

Correct Answer: 3

Explanation:

- **Prevent**: A coordinated Patch Management System with Automated Patching removes the Struts vulnerability.
- **Detect**: Behavioral Analytics layered with a properly configured SIEM spots lateral SSH pivot attempts.
- **Prevent Access**: Micro-segmentation plus WAF blocks east-west traffic from web to DB.
- **Respond**: SOAR-automated playbooks plus Incident Response Team (IRT) coordination ensure rapid containment.

Question: Equifax Lateral Movement & Exfiltration

Scenario

Same 2017 Equifax AWS layout: public Struts EC2 → ELB → Route 53; private RDS PostgreSQL; no micro-segmentation; disabled GuardDuty; misconfigured Inspector; CloudWatch Logs→SIEM. After initial compromise, attackers used SSH to pivot across subnets, harvested PII files, and exfiltrated them via an encrypted tunnel to an external C2 server.

Tactics Used

- **Lateral Movement (T1021.002)**: Remote Services
- **Collection (T1005)**: Data from Local System
- **Exfiltration (T1041)**: Exfiltration Over C2

Security Controls

- Network Segmentation / Micro-segmentation
- Encrypted Data in Transit (TLS east-west)
- Behavioral Analytics (UEBA)
- SOAR Playbooks
- SIEM with CloudWatch Logs
- IDS with valid certificates

Question

Which combination of controls best blocks lateral pivot, prevents data harvesting, and responds to exfiltration in this scenario?

Options

1.

- **Prevent (ZTA)**: Micro-segmentation + Encrypted Data in Transit
- **Detect (ASA)**: IDS with valid certificates + Behavioral Analytics
- **Prevent Access (DiD)**: Network Segmentation + WAF

- **Respond (ASA)**: SOAR Playbooks + SIEM Alerts

2.

- **Prevent (ZTA)**: Micro-segmentation + Encrypted Data in Transit
- **Detect (ASA)**: Behavioral Analytics + SIEM with CloudWatch Logs
- **Prevent Access (DiD)**: Network Segmentation + IDS with valid certificates
- **Respond (ASA)**: SOAR Playbooks + IRT Coordination

3.

- **Prevent (ZTA)**: Micro-segmentation + WAF
- **Detect (ASA)**: Behavioral Analytics + SIEM with CloudWatch Logs
- **Prevent Access (DiD)**: Encrypted Data in Transit + Network Segmentation
- **Respond (ASA)**: SOAR Playbooks + IRT Coordination

4.

- **Prevent (DiD)**: Network Segmentation + IDS with valid certificates
- **Detect (ASA)**: Behavioral Analytics + SIEM Alerts
- **Prevent Access (ZTA)**: Micro-segmentation + Encrypted Data in Transit
- **Respond (ASA)**: SOAR Playbooks + Certificate Management

5.

- **Prevent (DiD)**: WAF + Certificate Management
- **Detect (ASA)**: IDS with valid certificates + Automated Patching
- **Prevent Access (ZTA)**: Encrypted Data in Transit + WAF
- **Respond (ASA)**: IRT Coordination + Behavioral Analytics

6.

- **Prevent (DiD)**: WAF + Automated Patching
- **Detect (ASA)**: CloudWatch Logs + Behavioral Analytics
- **Prevent Access (ZTA)**: Micro-segmentation + Certificate Management

- **Respond (ASA)**: IRT Coordination + SOAR Playbooks

Correct Answer: 6

Explanation:

- **Prevent**: Micro-segmentation plus TLS east-west stops pivot and harvest.
- **Detect**: UEBA-style Behavioral Analytics with SIEM alerts reveal unusual DB access.
- **Prevent Access**: Traditional network segmentation reinforced by IDS coverage halts unauthorized connections.
- **Respond**: Automated SOAR playbooks with IRT coordination enable rapid lock-down and forensic capture.

Question: SolarWinds Supply-Chain Compromise

Scenario

In 2020, a vendor's build pipeline ran in an AWS VPC with three subnets: **Build** (EC2 build servers), **Update** (EC2 distribution servers), and **Client** (EC2 customer appliances). Build artifacts were compiled, cryptographically signed, then pushed to the Update servers. Route 53 DNS directed clients to updates. Amazon CloudTrail recorded API and file events. No tamper detection or continuous validation was in place, and micro-segmentation between build and update subnets was minimal. Attackers injected a backdoor into the signed binaries, which were then automatically distributed to 18,000+ client systems.

Tactics Used

- **Initial Access (T1195.002)**: Supply Chain Compromise
- **Defense Evasion (T1553.002)**: Forge Code Signing
- **Persistence (T1543.003)**: Create or Modify System Process

- **Lateral Movement (T1021.002)**: Remote Services across subnets
- **Exfiltration (T1041)**: Exfiltration Over C2

Security Controls

- Code Signing Verification
- Network Segmentation / Micro-segmentation
- Tamper Detection
- Audit Logging (CloudTrail)
- Continuous Validation of Artifacts
- Behavioral Monitoring (UEBA)
- Least Privilege Access
- Dynamic Network Policies
- Encrypted Communication (mTLS)
- SOAR Playbooks (automated response)

Question

Which combination of controls best prevents backdoor injection, detects pipeline tampering, restricts unauthorized update distribution, and responds to malicious artifacts in this SolarWinds scenario?

Options

1.

- **Prevent (DiD)**: Code Signing Verification + Network Segmentation
- **Detect (ASA)**: Tamper Detection + Audit Logging
- **Prevent Access (ZTA)**: Continuous Validation + Micro-segmentation
- **Respond (ASA)**: Dynamic Network Policies + SOAR Playbooks

2.

- **Prevent (DiD)**: Tamper Detection + Continuous Validation
- **Detect (DiD)**: Audit Logging + Behavioral Monitoring
- **Prevent Access (ZTA)**: Least Privilege Access + Encrypted

Communication
- **Respond (ASA)**: SOAR Playbooks + CloudTrail Alerts

3.

- **Prevent (DiD)**: Code Signing Verification + Continuous Validation
- **Detect (ASA)**: Audit Logging + Anomaly Detection in Builds
- **Prevent Access (ZTA)**: Network Segmentation + Micro-segmentation
- **Respond (ASA)**: Dynamic Network Policies + Encrypted Data Channels

4.

- **Prevent (ASA)**: Behavioral Monitoring + Micro-segmentation
- **Detect (DiD)**: Tamper Detection + Audit Logging
- **Prevent Access (ZTA)**: Least Privilege Access + Continuous Validation
- **Respond (ASA)**: SOAR Playbooks + Certificate Management

5.

- **Prevent (DiD)**: Code Signing Verification + Network Segmentation
- **Detect (ASA)**: Audit Logging + Anomaly Detection in Builds
- **Prevent Access (ZTA)**: Continuous Validation + Least Privilege Access
- **Respond (ASA)**: Dynamic Network Policies + Encrypted Data Channels

6.

- **Prevent (DiD)**: Code Signing Verification + Tamper Detection
- **Detect (ASA)**: Behavioral Monitoring + Audit Logging
- **Prevent Access (ZTA)**: Encrypted Communication + Micro-segmentation
- **Respond (ASA)**: Dynamic Network Policies + IRT Coordination

Correct Answer: 5

Explanation:

- **Prevent**: Code Signing Verification combined with strong network segmentation blocks tampered builds.

- **Detect**: CloudTrail audit logs with in-pipeline anomaly detection catch unauthorized modifications.
- **Prevent Access**: Continuous Validation plus Least Privilege Access ensures only genuine, minimal-scope artifacts
deploy.
- **Respond**: Dynamic network policies and secure data channels enable rapid containment and rollback.

Question: SolarWinds Lateral Movement & Persistence

Scenario

The SolarWinds environment (build/update/client subnets, signed artifacts, no tamper checks) allowed attackers to pivot from build to update servers, install persistent backdoors in system services, and maintain access across multiple pipeline stages.

Tactics Used

- **Lateral Movement (T1021.002)**: Remote Services
- **Persistence (T1543.003)**: Create or Modify System Process

Security Controls

- Micro-segmentation
- Continuous Validation
- Tamper Detection
- Least Privilege Access
- Behavioral Monitoring
- Automated Integrity Checks
- Encrypted Communication

Question

Which combination of controls best halts lateral movement in the pipeline and prevents installation of persistent
backdoors?

Options

1.

- **Prevent (DiD)**: Network Segmentation + Automated Integrity Checks
- **Detect (ASA)**: Behavioral Monitoring + Tamper Detection
- **Prevent Access (ZTA)**: Micro-segmentation + Continuous Validation
- **Respond (ASA)**: SOAR Playbooks + Audit Logging

2.

- **Prevent (DiD)**: Tamper Detection + Code Signing Verification
- **Detect (ASA)**: Audit Logging + Behavioral Monitoring
- **Prevent Access (ZTA)**: Micro-segmentation + Least Privilege Access
- **Respond (ASA)**: Dynamic Network Policies + Encrypted Data Channels

3.

- **Prevent (ZTA)**: Continuous Validation + Micro-segmentation
- **Detect (ASA)**: Behavioral Monitoring + Anomaly Detection in Builds
- **Prevent Access (DiD)**: Automated Integrity Checks + Network Segmentation
- **Respond (ASA)**: SOAR Playbooks + IRT Coordination

4.

- **Prevent (DiD)**: Network Segmentation + Code Signing Verification
- **Detect (ASA)**: Tamper Detection + Audit Logging
- **Prevent Access (ZTA)**: Continuous Validation + Least Privilege Access
- **Respond (ASA)**: Dynamic Network Policies + Behavioral Monitoring

5.

- **Prevent (ASA)**: Behavioral Monitoring + Continuous Validation
- **Detect (DiD)**: Tamper Detection + Automated Integrity Checks
- **Prevent Access (ZTA)**: Micro-segmentation + Encrypted Communication
- **Respond (ASA)**: SOAR Playbooks + Audit Logging

6.

- **Prevent (DiD)**: Code Signing Verification + Network Segmentation
- **Detect (ASA)**: Tamper Detection + Behavioral Monitoring
- **Prevent Access (ZTA)**: Micro-segmentation + Continuous Validation
- **Respond (ASA)**: SOAR Playbooks + Dynamic Network Policies

Correct Answer: 6

Explanation:

- **Prevent**: Code Signing Verification plus network segmentation blocks pivot to update tier.
- **Detect**: Tamper Detection and Behavioral Monitoring flag unauthorized backdoor writes.
- **Prevent Access**: Micro-segmentation with continuous artifact validation stops execution of unapproved binaries.
- **Respond**: Automated playbooks and dynamic policies isolate compromised nodes.

Question: Capital One SSRF & Exfiltration

Scenario

In 2019, a financial firm's AWS VPC hosted an EC2 web application in a public subnet behind AWS WAF, accessible via Route 53. The EC2 had an IAM role granting wide S3 bucket access. WAF rules were too permissive, allowing an SSRF payload to call the EC2 metadata service, returning temporary credentials. Attackers used these credentials to enumerate and download data from 100+ S3 buckets, then exfiltrated it to an external C2. CloudWatch Logs and GuardDuty were active, but IAM policies were overly broad and micro-segmentation was absent.

Tactics Used

- **Initial Access (T1190)**: Exploit Public-Facing Application (SSRF)
- **Defense Evasion (T1098)**: Account Manipulation (metadata token reuse)
- **Discovery (T1083)**: File and Directory Discovery (S3 enumeration)
- **Collection (T1005)**: Data from Local System (S3 objects)
- **Exfiltration (T1041)**: Exfiltration Over C2

Security Controls

- WAF Rule Validation
- IAM Role Hardening (Least Privilege)
- Network Segmentation / Micro-segmentation
- Monitoring & Logging (CloudWatch + GuardDuty)
- Data Encryption at Rest (S3 SSE-KMS)
- Continuous Verification (ZTA per-request auth)
- Adaptive WAF Rules (real-time tuning)
- Automated IAM Adjustment (revoke tokens)
- Dynamic Monitoring
- Encrypted Data Channels

Question

Which combination of controls best prevents SSRF, detects token misuse, restricts S3 access, and responds to
exfiltration?

Options

1.

- **Prevent (DiD)**: WAF Rule Validation + IAM Role Hardening
- **Detect (ASA)**: Monitoring & Logging + Behavioral Monitoring
- **Prevent Access (ZTA)**: Continuous Verification + Network Segmentation
- **Respond (ASA)**: Automated IAM Adjustment + Dynamic Monitoring

2.

- **Prevent (DiD)**: IAM Role Hardening + Data Encryption at Rest
- **Detect (ASA)**: GuardDuty + CloudWatch Logs
- **Prevent Access (ZTA)**: Micro-segmentation + Continuous Verification
- **Respond (ASA)**: Adaptive WAF Rules + Encrypted Data Channels

3.

- **Prevent (ZTA)**: Continuous Verification + WAF Rule Validation
- **Detect (ASA)**: GuardDuty + Behavioral Monitoring
- **Prevent Access (DiD)**: Network Segmentation + IAM Role Hardening
- **Respond (ASA)**: Automated IAM Adjustment + Adaptive WAF Rules

4.

- **Prevent (DiD)**: WAF Rule Validation + Continuous Verification
- **Detect (ASA)**: Behavioral Monitoring + CloudWatch Logs
- **Prevent Access (ZTA)**: IAM Role Hardening + Micro-segmentation
- **Respond (ASA)**: Dynamic Monitoring + Encrypted Data Channels

5.

- **Prevent (ASA)**: Adaptive WAF Rules + Continuous Verification
- **Detect (DiD)**: GuardDuty + Monitoring & Logging
- **Prevent Access (ZTA)**: IAM Role Hardening + Network Segmentation
- **Respond (ASA)**: Automated IAM Adjustment + Certificate Management

6.

- **Prevent (DiD)**: WAF Rule Validation + IAM Role Hardening

- **Detect (ASA)**: GuardDuty + Behavioral Monitoring
- **Prevent Access (ZTA)**: Continuous Verification + Micro-segmentation
- **Respond (ASA)**: Automated IAM Adjustment + Adaptive WAF Rules

Correct Answer: 6

Explanation:

- **Prevent**: A strict WAF and least-privilege IAM role stop SSRF and credential overreach.
- **Detect**: GuardDuty plus UEBA-style Behavioral Monitoring catch metadata misuse.
- **Prevent Access**: Per-request Continuous Verification and micro-segmentation block unauthorized S3 calls.
- **Respond**: Automated IAM revocation and adaptive WAF tuning throttle and lock down exfiltration channels.

TRUE or FALSE

Question: The Linux kernel's Completely Fair Scheduler (CFS) guarantees that a process pinned to a specific CPU core will never be preempted by another process, even if the latter has a higher priority, due to the per-CPU runqueue design. (False)

The Completely Fair Scheduler (CFS) is the default process scheduler in the Linux kernel, designed to fairly allocate CPU time among running processes. While it uses a per-CPU runqueue design—where each CPU core maintains its own queue of runnable processes—this does not mean a process pinned to a specific core is immune to preemption.

- **Per-CPU Runqueues**: Each core manages its own runqueue to reduce contention and improve scalability. A process pinned to a core (via CPU affinity) will only run on that core.
- **Preemption**: CFS allows preemption based on priority and fairness. It

uses a "virtual runtime" metric to track how much CPU time a process has consumed. If a higher-priority process (one with less virtual runtime or an explicitly higher priority) becomes runnable on the same core, it can preempt the current process.

- **Pinning Limitation**: Pinning restricts a process to a specific core but doesn’t disable the scheduler’s ability to switch between processes on that core.

Thus, the statement is **false**: CFS does not guarantee that a pinned process will never be preempted by a higher-priority process on the same core.

Question 2: In containerized environments using overlayfs, a write operation to a file in the lower layer directly modifies the original file, bypassing the upper layer, unless explicitly configured otherwise. (False)

Overlayfs is a union filesystem commonly used in containerized environments (e.g., Docker) to manage layered storage. It combines a read-only **lower layer** (the base image) and a writable **upper layer** (for container-specific changes) into a single merged view.

- **How Writes Work**: When a file in the lower layer is modified, overlayfs employs a copy-on-write (CoW) mechanism. The original file is copied to the upper layer, and the write operation modifies this copy, leaving the lower layer intact.
- **Read-Only Lower Layer**: The lower layer is inherently read-only in typical container setups, ensuring the base image remains unchanged for reuse across multiple containers.
- **No Direct Modification**: There’s no default behavior or configuration in overlayfs that allows writes to directly alter the lower layer, bypassing the upper layer.

The statement is **false**: Write operations do not modify the original file in the lower layer; changes are always applied to the upper layer via CoW.

Question 3: The Linux kernel’s handling of Non-Maskable Interrupts (NMIs) allows them to be deferred to a later execution context, such as a softirq, to avoid disrupting critical kernel operations. (False)

Non-Maskable Interrupts (NMIs) are critical interrupts in the Linux kernel, typically triggered by severe hardware events (e.g., memory errors or watchdog timeouts), and they differ from regular interrupts in key ways.

- **Immediate Execution**: NMIs cannot be masked or ignored by the CPU. When an NMI occurs, it interrupts the current execution—whether in user space or kernel space, even within critical sections—and its handler runs immediately.
- **No Deferral Mechanism**: Unlike regular interrupts, which can be deferred to softer contexts like softirqs or tasklets, NMIs lack a deferral mechanism. Their handlers must execute in the interrupt context right away.
- **Design Implication**: Because NMIs can disrupt critical operations, their handlers are kept minimal and carefully written to avoid instability.

The statement is **false**: NMIs cannot be deferred to a later context like a softirq; they are handled immediately.
