Ionita Alexandru-Mihail
1241B

UNSTPB
FILS

Security & Encryption

Homework

Our Case: We start working for a healthcare provider that operates with digital patient records, online appointments & telehealth services, among others

1. Transitioning from a monolithic architecture to microservices

In the event we need to scale up specific services of the company, such as the online appointments system, we would have to affect the whole system. Transitioning to a microservices architecture would help us separate the different services, which would communicate with each other through secure API gateways. This approach also makes it easier to patch or update individual components, limits destruction in care of an attacker breaching the system and enforces a zero-trust model.

2. Securing the databases (SQL & NoSQL)
Given the sensitive nature of the data handled by the company, such as patient records and billing information, the databases must be secured on multiple levels. For SQL databases, we would implement parameterized queries and strong authentication policies to prevent SQL injection and unauthorized access. For NoSQL databases, we would ensure all inputs are validated properly and access controls are enforced. Encrypting data both at rest and in transit would further minimize the risk of data leaks.

3. Improving API security
Since our APIs are the bridge between internal services and third-party systems, they are a high-risk point of entry. To protect them, we would use OAuth 2.0 for access control, validate and sanitize all inputs, and implement rate limiting to prevent abuse or denial-of-service attacks. Encrypting all API traffic and monitoring logs would help us detect suspicious activity and maintain a secure communication channel across the system.

4. Securing the caching layer
A caching system based on Redis would improve performance by storing frequently accessed data, but it could also be a weak link if not properly secured. To address this, we would apply access restrictions, use TLS encryption for communication, and ensure that sensitive data is either not cached or set to expire quickly. This would help us prevent cache poisoning, unauthorized access, and the risk of serving outdated or incorrect information to users.

5. Hardening the load balancer (Nginx)
The load balancer plays a key role in routing user traffic and ensuring service availability, which makes it a valuable target for attackers. We would restrict access to its admin interface using firewalls and strong authentication, enforce HTTPS with modern TLS configurations, and apply

security headers to defend against injection or manipulation attempts. Rate limiting and tools like fail2ban would help us mitigate denial-of-service attacks and keep the service stable.