1. Top 2: Cryptographic Failures

Description

The first thing is to determine the protection needs of data in transit and at rest. For example, passwords, credit card numbers, health records, personal information, and business secrets require extra protection, mainly if that data falls under privacy laws, e.g., EU's General Data Protection Regulation (GDPR), or regulations, e.g., financial data protection such as PCI Data Security Standard (PCI DSS). For all such data:

- Is any data transmitted in **clear text?** This concerns protocols such as HTTP, SMTP, FTP also using TLS upgrades like STARTTLS. **External internet traffic** is hazardous(). Verify all internal traffic, e.g., between load balancers, web servers, or back-end systems.
- Are any **old or weak cryptographic** algorithms or protocols used either by default or in older code?
- Are default **crypto keys** in use, weak crypto keys generated or re-used, or is proper key management or rotation missing? Are crypto keys checked into source code repositories?
- Is encryption not enforced, e.g., are any **HTTP headers (browser) security directives(X-Type Headers)** or headers missing?
- Is the received **server certificate and the trust chain** properly validated?
- Are **initialization vectors ignored, reused, or not generated sufficiently secure** for the cryptographic mode of operation? Is an insecure mode of operation such as ECB in use? Is encryption used when authenticated encryption is more appropriate?

  - **Initializing Vector (IV)**, It ensures that even if you encrypt the same plaintext multiple times with the same key, the **resulting ciphertexts will be different.**

  - ECB stands for "Electronic Codebook." It is one of the simplest modes of operation for symmetric block ciphers like the Advanced Encryption Standard (AES). In ECB mode, each block of plaintext is independently encrypted with the same encryption key. This means that identical plaintext blocks will produce identical ciphertext blocks.
- Are **passwords being used as cryptographic keys in absence of a password base key** derivation function?
- Are deprecated hash functions such as **MD5 or SHA1** in use, or are non-cryptographic hash functions used when cryptographic hash functions are needed?
- Are deprecated **cryptographic padding methods** such as **PKCS number 1 v1.5 in use**?
  In AES Encryption, every block of cypher text would be 16 bytes. So, If any block generated less number of that extra bytes will be added. Example: If original block size is 13 bytes then it will add extra 3 bytes.

  Original Message: [M1 M2 M3 M4 M5 M6 M7 M8 M9 M10 M11 M12 M13]
  After PKCS#7 Padding: [M1 M2 M3 M4 M5 M6 M7 M8 M9 M10 M11 M12 M13 03 03 03]

  During Decryption it will check last digit. If it get 3, then it will remove last 3 bytes.

How to Prevent

Do the following, at a minimum, and consult the references:

- Make sure to encrypt all sensitive data at rest.
- Ensure up-to-date and strong standard algorithms, protocols, and keys are in place; use proper key management.
- **Disable caching** for response that contain sensitive data.
- Do not use legacy protocols such as FTP and SMTP for transporting sensitive data. Use **HTTPS.**
- Store passwords using strong adaptive and **salted hashing functions with a work factor (delay factor), such as Argon2, scrypt, bcrypt or PBKDF2**. -it make delay intentionally decrypt or match the password.
- Always use authenticated encryption(SSL Wireless a well known Certificate Authority in BD) instead of just encryption.
- Keys should be generated cryptographically randomly and stored in memory**(not DB)** as byte arrays. If a password is used, then it must be converted to a key via an appropriate password base key derivation function.
- Avoid deprecated cryptographic functions and padding schemes, such as **MD5, SHA1, PKCS number 1 v1.5 .**

Example Attack Scenarios

Scenario #1: An application encrypts credit card numbers in a database using automatic database encryption. However, this data is automatically decrypted when retrieved, allowing a SQL injection flaw to retrieve credit card numbers in clear text. (Standalone database is secure in this case. But application is getting clear text. So, attacker will try to injection using application and got clear text).

Scenario #2: A site doesn't use or enforce TLS for **all pages** or supports weak encryption. An attacker monitors network traffic (e.g., at an insecure wireless network), downgrades connections from HTTPS to HTTP, intercepts requests, and steals the user's session cookie. The attacker then replays this cookie and hijacks the user's (authenticated) session, accessing or modifying the user's private data. Instead of the above they could alter all transported data, e.g., the recipient of a money transfer.

Scenario #3: The password database **uses unsalted or simple hashes to store everyone's passwords**. A file upload flaw allows an attacker to retrieve the password database. All the unsalted hashes can be exposed with a rainbow table of pre-calculated hashes. Hashes generated by simple or fast hash functions may be cracked by GPUs, even if they were salted.

- Top 2: Hands On