M5: Insecure n

Threat Agent

Application Specific

Most modern mobile ata
with one or more rem ta
transmission takes p ough
the mobile device's c ernet,
a threat agent listeni ept and
modify the data if it t using
a deprecated encrypt its
might have different
sensitive information identity
theft and more. The f xist:

- An adversary tha ork
  (compromised o
- Rogue carrier or , cell
  towers, proxy's,
- Malware on your mobile device.

Attack Vectors

Exploitability EASY

While modern applications do reply on cryptographic protocols such as SSL/TLS, they can sometimes have flaws in their implementations like:

The OWASP® Foundation works to improve the security of software through its community-led open source software projects, hundreds of chapters worldwide, tens of thousands of members, and by hosting local and global conferences.

Upcoming OWASP Global Events

OWASP Global AppSec EU 2026 - Vienna, Austria

- June 22-26, 2026

OWASP Global AppSec USA 2026 - San Francisco, CA

- November 2-6, 2026

- Using deprecated protocols and/or bad configuration settings;
- Accepting bad ssl certificates (self-signed, revoked, expired, wrong host…); or
- Inconsistency (having SSL/TLS only on select workflows such as authentication).

Security Weakness

Prevalence COMMON

Detectability AVERAGE

While modern mobile applications aim to protect network traffic, they often have inconsistencies in their implementation. These inconsistencies can lead to vulnerabilities that expose data and session IDs to interception. Just because an app uses transport security protocols doesn’t mean it’s implemented correctly. To identify basic flaws, you can observe the network traffic on the phone. However, detecting more subtle flaws requires a closer look at the application’s design and configuration.

Technical Impacts

Impact SEVERE

This flaw can expose user data which might lead to account takeover, user impersonation, PII data leaks and more, for instance an attacker might intercept user credentials, session, 2FA tokens which can open the door for more elaborate attacks.

Business Impacts

Impact MODERATE

At a minimum, interception of sensitive data through a communication channel will result in a privacy violation.

The violation of a user’s confidentiality may result in:

- Identity theft;
- Fraud, or
- Reputational Damage.

Am I Vulnerable To ‘Insecure Communication’?

This risk covers all aspects of getting data from point A to point B, but doing it insecurely. It encompasses mobile-to-mobile communications, app-to-server communications, or mobile-to-something-else communications. This risk includes all communications technologies that a mobile device might use: TCP/IP, WiFi, Bluetooth/Bluetooth-LE, NFC, audio, infrared, GSM, 3G, SMS, etc.

All the TLS communications issues go here. All the NFC, Bluetooth, and WiFi issues go here.

The prominent characteristics include packaging up some kind of sensitive data and transmitting it into or out of the device. Some examples of sensitive data include encryption keys, passwords, private user information, account details, session tokens, documents, metadata, and binaries. The sensitive data can be coming to the device from a server, it can be coming from an app out to a server, or it might be going between the device and something else local (e.g., an NFC terminal or NFC card). The defining characteristic of this risk is the existence of two devices and some data passing between them.

If the data is being stored locally in the device itself, that’s #Insecure Data. If the session details are communicated securely (e.g., via a strong TLS connection) but the session identifer itself is bad (perhaps it is predictable, low entropy, etc.), then that’s an #Insecure Authentication problem, not a communication problem.

The usual risks of insecure communication are around data integrity, data confidentiality, and origin integrity. If the data can be changed while in transit, without the change being detectable (e.g., via a man-in-the-middle attack) then that is a good example of this risk. If confidential data can be exposed, learned, or derived by observing the communications as it happens (i.e., eavesdropping) or by recording the conversation as it happens and attacking it later (offline attack), that’s also an insecure communication problem. Failing to properly setup and validate a TLS connection (e.g., certificate checking, weak ciphers, other TLS configuration problems) are all here in insecure communication.

How Do I Prevent ‘Insecure Communication’?

General Best Practices

- Assume that the network layer is not secure and is susceptible to eavesdropping.
- Apply SSL/TLS to transport channels that the mobile app will use to transmit data to a backend API or web service.
- Account for outside entities like third-party analytics companies, social networks, etc. by using their SSL versions when an application runs a routine via the browser/webkit. Avoid

mixed SSL sessions as they may expose the user’s session ID.
- Use strong, industry standard cipher suites with appropriate key lengths.
- Use certificates signed by a trusted CA provider.
- Never allow bad certificates (self-signed, expired, untrusted root, revoked, wrong host..).
- Consider certificate pinning.
- Always require SSL chain verification.
- Only establish a secure connection after verifying the identity of the endpoint server using trusted certificates in the key chain.
- Alert users through the UI if the mobile app detects an invalid certificate.
- Do not send sensitive data over alternate channels (e.g, SMS, MMS, or notifications).
- If possible, apply a separate layer of encryption to any sensitive data before it is given to the SSL channel. In the event that future vulnerabilities are discovered in the SSL implementation, the encrypted data will provide a secondary defense against confidentiality violation.
- During development cycles, avoid overriding SSL verification methods to allow untrusted certificates, instead try using self-signed certificates or a local development certificate authority (CA)
- During security assessments, it is advised to analyze application traffic to see if any traffic goes through plaintext channels

iOS Specific Best Practices

Default classes in the latest version of iOS handle SSL cipher strength negotiation very well. Trouble comes when developers temporarily add code to bypass these defaults to accommodate development hurdles. In addition to the above general practices:

- Ensure that certificates are valid and fail closed.
- When using `CFNetwork`, consider using the Secure Transport API to designate trusted client certificates. In almost all situations, `NSStreamSocketSecurityLevelTLSv1` should be used for higher standard cipher strength.
- After development, ensure all `NSURL` calls (or wrappers of `NSURL`) do not allow self signed or invalid certificates such as the `NSURL` class method `setAllowsAnyHTTPSCertificate`.
- Consider using certificate pinning by doing the following: export your certificate, include it in your app bundle, and anchor it to your trust object. Using the NSURL method `connection:willSendRequestForAuthenticationChallenge`: will now accept your cert.

Android Specific Best Practices

- Remove all code after the development cycle that may allow the application to accept all certificates such as org.apache.http.conn.ssl.AllowAllHostnameVerifier or SSLSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER. These are equivalent to trusting all certificates.
- If using a class which extends SSLSocketFactory, make sure checkServerTrusted method is properly implemented so that server certificate is correctly checked.

- **Avoid overriding `onReceivedSslError` to allow invalid SSL certificates**

Example Attack Scenarios

There are a few common scenarios that penetration testers frequently discover when inspecting a mobile app’s communication security:

Lack of certificate inspection

The mobile app and an endpoint successfully connect and perform a TLS handshake to establish a secure channel. However, the mobile app fails to inspect the certificate offered by the server and the mobile app unconditionally accepts any certificate offered to it by the server. This destroys any mutual authentication capability between the mobile app and the endpoint. The mobile app is susceptible to man-in-the-middle attacks through a TLS proxy.

Weak handshake negotiation

The mobile app and an endpoint successfully connect and negotiate a cipher suite as part of the connection handshake. The client successfully negotiates with the server to use a weak cipher suite that results in weak encryption that can be easily decrypted by the adversary. This jeopardizes the confidentiality of the channel between the mobile app and the endpoint.

Privacy information leakage

The mobile app transmits personally identifiable information to an endpoint via non-secure channels instead of over SSL/TLS. This jeopardizes the confidentiality of any privacy-related data between the mobile app and the endpoint.

Credential information leakage

The mobile app transmits user credentials to an endpoint via non-secure channels instead of over SSL/TLS. This allows an adversary to intercept those credentials in cleartext.

Two-Factor authentication bypass

The mobile app receives a session identifier from an endpoint via non-secure channels instead of over SSL/TLS. This allows an adversary to bypass two-factor authentication by using the intercepted session identifier.

References

- OWASP
  - OWASP
- External
  - External References

Edit on GitHub

Red Hat Scitum TELMEX está contigo ATLASSIAN

Adobe CORGEA OSec

Bloomberg® Fortify BIONIC

Become a corporate supporter

HOME PROJECTS CHAPTERS EVENTS ABOUT

PRIVACY SITEMAP CONTACT