M3: Insecure Communication

Threat Agents

Application Specific

When designing a mobile application, data is commonly exchanged in a client-server fashion. When the solution transmits its data, it must traverse the mobile device's carrier network and the internet. Threat agents might exploit vulnerabilities to intercept sensitive data while it's traveling across the wire. The following threat agents exist:

- An adversary that shares your local network (compromised or monitored Wi-Fi);
- Carrier or network devices (routers, cell towers, proxy's, etc); or
- Malware on your mobile device.

Attack Vectors

Exploitability EASY

The exploitabilty factor of monitoring a network for insecure communications ranges. Monitoring traffic over a carrier's network is harder than that of monitoring a local coffee shop's traffic. In general, targeted attacks are easier to perform.

Security Weakness

Prevalence COMMON
Detectability AVERAGE

Mobile applications frequently do not protect network traffic. They may use SSL/TLS during authentication but not elsewhere. This inconsistency leads to the risk of exposing data and session IDs to interception. The use of transport security does not mean the app has implemented it correctly. To detect basic flaws, observe the phone's network traffic. More subtle flaws require inspecting the design of the application and the applications configuration.

Technical Impacts

Impact SEVERE

This flaw exposes an individual user's data and can lead to account theft. If the adversary intercepts an admin account, the entire site could be exposed. Poor SSL setup can also facilitate phishing and MITM attacks.

Business Impacts

Application / Business Specific

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 identifier 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 sensitive information, session tokens, or other sensitive 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 self-signed certificates, and consider certificate pinning for security conscious applications.
- 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.

Newer threats allow an adversary to eavesdrop on sensitive traffic by intercepting the traffic within the mobile device just before the mobile device's SSL library encrypts and transmits the network traffic to the destination server. See M10 for more information on the nature of this risk.

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.

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. This jeopardizes the confidentiality of any privacy-related data between the mobile app and the endpoint.

References

- OWASP
  - OWASP
- External
  - External References

Edit on GitHub