Guide to Network Security Configuration in Android P

Published: 2018-08-15 · Archived: 2026-04-05 17:54:55 UTC

As data privacy becomes increasingly important, Google has introduced mobile OS enhancements to safeguard all data that traverses Android mobile devices and endpoints. Set for release in August 2018, network communications in Android 9.0 P (Pie) will default to TLS. Android mobile app developers will either need to update their back-end services to support HTTPS or implement the Android Network Security Configuration feature to prevent app connections from failing.

When Android 6.0 Marshmallow was released, Google introduced the manifest attribute `android:usesCleartextTraffic` as a means to protect against accidental use of cleartext traffic. Android 7.0 Nougat extended this attribute by introducing the Android Network Security Configuration feature, which allows developers to be more prescriptive about secure communications. Network Security Configuration is an XML file in which developers customize network security settings for an Android app.

Some of you may think this sounds familiar. iOS uses a similar client side check known as App Transport Security. While there are quite a few similarities in what protections Network Security Configuration offers when compared to NSAppTranportSecurity, the two take very different approaches to network security on their individual platforms. To learn more about ATS, check out my previous blog.

Let's examine several benefits of using Network Security Configuration in Android mobile apps and dive into best practices for implementing this feature.

1. Protect against regressions to cleartext traffic

Security is more about layers of protection than a single iron wall. The Android Network Security Configuration feature provides a simple layer to protect apps from unintentionally transmitting sensitive data in unencrypted cleartext.

If you don't know what "unencrypted communications" means, think of it this way — let's say your office has a policy to send all shipments via UPS. A new intern joins the office and is tasked with shipping equipment to an office across the country. Oblivious to the policy and with all the best intentions, the intern sets up all shipments to be sent through an unknown, less expensive service. The Android Network Security Configuration feature is like the shipping/receiving manager who examines all inbound and outbound shipments and stops the shipment before the equipment gets into the hands of an unvetted delivery system. It can be used to prevent the accidental use of untrusted, unencrypted connections.

One of the biggest changes in Android 9 is that `cleartextTrafficPermitted` is set to `false` by default. This means that if you don't see this flag explicitly set to false, and the app is targeting API levels lower than 28, the flag will be honored as true.

Another capability of the `cleartextTrafficPermitted` flag being used in the Network Security Config is the ability to enforce the `true` setting on specific domains and subdomains:

Network Security Configuration

```
<network-security-config>
    <domain-config cleartextTrafficPermitted="true">
        <domain includeSubdomains="true">insecure.example.com</domain>
    </domain-config>
    <base-config cleartextTrafficPermitted="false" />
</network-security-config>
```

2. Set up trusted Certificate Authorities for secure connections

Trusted Certificate Authorities (CA) act as the circle of trust. In the previous example, the office policy was to ship with UPS, but that policy could be expanded to FedEx, DHL, and so forth. Essentially, who do you trust to securely send app data and prevent man-in-the-middle attacks? Developers can use the Android Network Security Configuration feature to designate which CAs they trust to issue certificates and ensure secure communications.

To start, Android Network Security Configuration gives developers a few options in terms of what CAs they should be trusting. By default, the trust anchor used by Android 7+ (Nougat, Oreo and Pie) will be the pre-installed system CA certificates, noted as `system`:

Trust Anchors: Android 7+

```
<network-security-config>
    <base-config>
        <trust-anchors>
            <certificates src="system"/>
        </trust-anchors>
    </base-config>
</network-security-config>
```

In Android 6 (Marshmallow) and below, your default `trust-anchor` will also include user installed certificate, noted as `user`:

Trust Anchors: Android 6 and below

```
<network-security-config>
    <base-config>
        <trust-anchors>
            <certificates src="system"/>
            <certificates src="user"/>
        </trust-anchors>
```

```
    </base-config>
</network-security-config>
```

Finally, you can set a custom trust anchor:

Trust Anchors: Custom

```
<network-security-config>
    <base-config>
        <trust-anchors>
            <certificates src="@raw/my_custom_ca"/>
        </trust-anchors>
    </base-config>
</network-security-config>
```

3. Implement certificate pinning

Implementing certificate pinning offers yet another layer of security. Let's revisit the running example of shipping equipment. If trusted CAs are like UPS, FedEx, etc., then certificate pinning is akin to specifying which of those companies' drivers you trust to send your shipment. The Android Network Security Configuration feature can be used to restrict communications with only specific certificates issued by a trusted CA.

We discussed different implementations of certificate pinning in a previous blog post. In the example below, we see we can pin to a specific domain and subdomains, set pins along with backups, and set an expiration date.

Certificate Pinning with Network Security Config

```
<network-security-config>
    <domain-config>
        <domain includeSubdomains="true">example.com</domain>
        <pin-set expiration="2018-01-01">
            <pin digest="SHA-256">7HIpactkIAq2Y49orFOOQKurWxmmSFZhBCoQYcRhJ3Y=</pin>
            <!-- backup pin -->
            <pin digest="SHA-256">fwza0LRMXouZHRC8Ei+4PyuldPDcf3UKgO/04cDM1oE=</pin>
        </pin-set>
    </domain-config>
</network-security-config>
```

4. Debug app network connections

Another option offered in Android Network Security Configuration is `debug-overrides`. This feature allows you to have settings in the Network Security Config that will only be usable when `android:debuggable` is set to `true`. For example, you can configure a custom `trust-anchor` for a quality assurance/pre-production

environment using a custom CA. This eases testing in a closed environment because the app store does not accept apps marked debuggable.

Debug Overrides

```
<network-security-config>
    <debug-overrides>
        <trust-anchors>
            <certificates src="@raw/debug_cas"/>
        </trust-anchors>
    </debug-overrides>
</network-security-config>
```

Implementation Advice

Now that you understand a few of the benefits of deploying Network Security Configuration, let's cover some best practices for implementing this file.

First, check the app manifest to see if it uses this feature. Look for the attribute `android:networkSecurityConfig`, which would appear similar to this:

Sample Manifest with Network Security Config

```
<manifest ... >
    <application android:networkSecurityConfig="@xml/network_security_config"
                    ... >
        ...
    </application>
</manifest>
```

Once you've located the Network Security Configuration file, it's time to check how to permit cleartext traffic.

The example code snippet below shows a bad example of how someone might use Network Security Configuration.

Bad Example: cleartext Permissions with Network Security Config

```
<network-security-config>
    <domain-config cleartextTrafficPermitted="false">
        <domain includeSubdomains="true">example.com</domain>
        <domain includeSubdomains="true">cdn.example2.com</domain>
    </domain-config>
    <base-config cleartextTrafficPermitted="true" />
</network-security-config>
```

While this ensures all traffic to `example.com` and `cdn.example2.com` is sent over HTTPS, the default configuration for all traffic sent to other domains can be cleartext. This completely defeats the intended purpose of the Network Security Configuration feature — to improve the privacy of all data transmitted through Android devices.

If your mobile app must send data in cleartext, then do this to only allow encrypted communications to certain domains. In addition, carefully scrutinize endpoints that HTTP is explicitly allowed to check for sensitive data and other API-related issues:

Recommended Implementation of cleartext Permissions

```
<network-security-config>
    <domain-config cleartextTrafficPermitted="true">
        <domain includeSubdomains="true">insecure.example.com</domain>
        <domain includeSubdomains="true">insecure.cdn.example.com</domain>
    </domain-config>
    <base-config cleartextTrafficPermitted="false" />
</network-security-config>
```

Another thing to consider regarding the `cleartextTrafficPermitted` flag is it defaults to `true` on Android 8 and lower. Because of this, explicitly set the flag to `false` in all apps' Network Security Configuration.

Let's look at another example below:

Bad Example: Default Implementation of Trust Anchors for Android 6 and below

```
<network-security-config>
    <base-config>
        <trust-anchors>
            <certificates src="system"/>
            <certificates src="user"/>
        </trust-anchors>
    </base-config>
</network-security-config>
```

In this example, we see the app is accepting `user` certs within the app's trust anchor. This means that user-installed certs will be trusted by the app the same way system pre-installed certs are. When configuring the `trust-anchors` used by the app, it's best to limit trust only to the `system` certs and, when necessary, to a custom CA built within the app. This can help prevent MITM attacks where an attacker is able to install a cert on the device. As part of the the protections introduced as part of Android 7, by default a user-installed certificate isn't trusted like pre-installed certificates. As stated earlier, Android 6 and below accept `user` certs by default, so it's important to explicitly select `system` and/or a custom CA when necessary and exclude `user` in all apps.

Recommended Implementation of Trust Anchors with Custom CA (When Necessary)

```
<network-security-config>
    <base-config>
        <trust-anchors>
            <certificates src="system"/>
            <certificates src="@raw/CustomCA"/>
        </trust-anchors>
    </base-config>
</network-security-config>
```

Let’s look at an example implementation of pinning:

Bad Example: Certificate Pinning with Network Security Config

```
<network-security-config>
    <domain-config>
        <domain includeSubdomains="true">example.com</domain>
        <pin-set>
            <pin digest="SHA-256">7HIpactkIAq2Y49orFOOQKurWxmmSFZhBCoQYcRhJ3Y=</pin>
        </pin-set>
    </domain-config>
</network-security-config>
```

The example above reveals two potential issues within the app. First, there is no expiration set for the `pin-set`. Second, there is no backup pin. A smart strategy is to set an expiration for certificates and have multiple backup pins. It is not unheard of to have four pins in rotation. If you do not explicitly set an expiration date for the pin, your app will fail to connect after expiration occurs. But if you set an expiration and the pin expires, the app will fail over to the system CA on the device instead of failing to connect.

Recommended Implementation of Certificate Pinning with Network Security Config

```
<network-security-config>
    <domain-config>
        <domain includeSubdomains="true">example.com</domain>
        <pin-set expiration="2018-01-01">
            <pin digest="SHA-256">7HIpactkIAq2Y49orFOOQKurWxmmSFZhBCoQYcRhJ3Y=</pin>
            <!-- backup pin -->
            <pin digest="SHA-256">fwza0LRMXouZHRC8Ei+4PyuldPDcf3UKgO/04cDM1oE=</pin>
        </pin-set>
    </domain-config>
</network-security-config>
```

Finally, let’s talk about the Network Security Config `debug-overrides`.

Recommended Implementation of Debug Overrides with Network Security Config

```
<network-security-config>
    <debug-overrides>
        <trust-anchors>
            <certificates src="@raw/debug_cas"/>
        </trust-anchors>
    </debug-overrides>
</network-security-config>
```

While the debug feature can help eliminate debug network code in an app, make sure to remove any debug CAs from the app. As a best practice, avoid leaving unnecessary information in your app that could present a security risk, especially if the internal CA certificates are included with final production build of the app.

Mobile Apps, IoT & 5G

WATCH LEADERSHIP ROUNDTABLE:

Powering Innovation and Performance Securely

From the perspective of a security analyst, knowing how to read and spot issues in the Network Security Configuration is important. As we can see, we are able to find some potential issues in the app before install. But keep in mind that these findings alone aren't proof that your app's network connections are secure.

You'll still need to determine if your app is performing hostname verification, because Network Security Configuration will not protect against those types of issues. Make sure your third-party libraries honor Network Security Configuration. If they don't, these protections may cause issues in your app. In addition, Network Security Configuration is not honored by lower-level network connections such as websockets. Finally, keep in mind that network-related issues in mobile are only a small part of the overall scope in mobile testing.

Overall, Android Network Security Configuration offers a lot of simple network security features for Android. If your app does not currently take advantage of Network Security Configuration, you will need to use it in the coming year.

Coinciding with the release of Android P, Google has begun enforcing target API levels in the app stores to reduce mobile OS fragmentation and push users to current releases. The current requirement is 26, although Google plans to incrementally increase that number with each new release. By this time next year, API Level 30 (Android Q) will probably be out. That means if your app uses HTTP, it must be declared in the Network Security Configuration file because the mandatory target API level will be 28 by then.

Stay abreast of other security enhancements by subscribing to our All Things Mobile DevSecOps newsletter and perform automated mobile application security testing to ensure your Android apps safely implement network communications.
