Network security configuration

The Network Security Configuration feature lets you customize your app's network security settings in a safe, declarative configuration file (https://developer.android.com/training/articles/security-config#FileFormat) without modifying app code. These settings can be configured for specific domains and for a specific app. The key capabilities of this feature are:

- **Custom trust anchors:** Customize which Certificate Authorities (CA) are trusted for an app's secure connections. For example, trusting particular self-signed certificates or restricting the set of public CAs that the app trusts.
- **Debug-only overrides:** Safely debug secure connections in an app without added risk to the installed base.
- **Cleartext traffic opt-out:** Protect apps from accidental usage of cleartext (unencrypted) traffic.
- **Certificate pinning:** Restrict an app's secure connection to particular certificates.

Add a Network Security Configuration file

The Network Security Configuration feature uses an XML file where you specify the settings for your app. You must include an entry in your app's manifest to point to this file. The following code excerpt from a manifest demonstrates how to create this entry:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:networkSecurityConfig="@xml/network_security_config"
                    ... >
        ...
    </application>
</manifest>
```

Customize trusted CAs

You might want your app to trust a custom set of CAs instead of the platform default. The most common reasons for this are:

- Connecting to a host with a custom CA, such as a CA that is self-signed or is issued internally within a company.
- Limiting the set of CAs to only the CAs you trust instead of every pre-installed CA.
- Trusting additional CAs not included in the system.

By default, secure connections (using protocols like TLS and HTTPS) from all apps trust the pre-installed system CAs, and apps targeting Android 6.0 (API level 23) and lower also trust the user-added CA store by default. You can customize your app's connections using `base-config` (for app-wide customization) or `domain-config` (for per-domain customization).

Configure a custom CA

You might want to connect to a host that uses a self-signed SSL certificate or to a host whose SSL certificate is issued by a non-public CA that you trust, such as your company's internal CA. The following code excerpt demonstrates how to configure your app for a custom CA in `res/xml/network_security_config.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config>
        <domain includeSubdomains="true">example.com</domain>
        <trust-anchors>
            <certificates src="@raw/my_ca"/>
        </trust-anchors>
    </domain-config>
</network-security-config>
```

Add the self-signed or non-public CA certificate, in PEM or DER format, to `res/raw/my_ca`.

Limit the set of trusted CAs

If you don't want your app to trust all CAs trusted by the system, you can instead specify a reduced set of CAs to trust. This protects the app from fraudulent certificates issued by any of the other CAs.

The configuration to limit the set of trusted CAs is similar to trusting a custom CA (#ConfigCustom) for a specific domain except that multiple CAs are provided in the resource. The following code excerpt demonstrates how to limit your app's set of trusted CAs in `res/xml/network_security_config.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config>
        <domain includeSubdomains="true">secure.example.com</domain>
        <domain includeSubdomains="true">cdn.example.com</domain>
        <trust-anchors>
            <certificates src="@raw/trusted_roots"/>
        </trust-anchors>
    </domain-config>
</network-security-config>
```

Add the trusted CAs, in PEM or DER format, to `res/raw/trusted_roots`. Note that if you use PEM format, the file must contain *only* PEM data and no extra text. You can also provide multiple `<certificates>` (#certificates) elements instead of one.

Trust additional CAs

You might want your app to trust additional CAs that aren't trusted by the system, such as if the system doesn't yet include the CA or the CA doesn't meet the requirements for inclusion in the Android system. You can specify multiple certificate sources for a configuration in `res/xml/network_security_config.xml` using code like the following excerpt.

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <base-config>
        <trust-anchors>
            <certificates src="@raw/extracas"/>
            <certificates src="system"/>
        </trust-anchors>
    </base-config>
</network-security-config>
```

Configure CAs for debugging

When debugging an app that connects over HTTPS, you may want to connect to a local development server that does not have the SSL certificate for your production server. To support this without any modification to your app's code, you can specify debug-only CAs, which are trusted *only* when android:debuggable (/guide/topics/manifest/application-element#debug) is `true`, by using `debug-overrides`. Normally, IDEs and build tools set this flag automatically for non-release builds.

This is safer than the usual conditional code because, as a security precaution, app stores do not accept apps that are marked debuggable.

The excerpt below shows how to specify debug-only CAs in `res/xml/network_security_config.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <debug-overrides>
        <trust-anchors>
            <certificates src="@raw/debug_cas"/>
        </trust-anchors>
    </debug-overrides>
</network-security-config>
```

Opt out of cleartext traffic

Note: The guidance in this section applies only to apps that target Android 8.1 (API level 27) or lower. Starting with Android 9 (API level 28), cleartext support is disabled by default.

If you intend for your app to connect to destinations using only secure connections, you can opt out of supporting cleartext (using the unencrypted HTTP protocol instead of HTTPS) to those destinations. This option helps prevent accidental regressions in apps due to changes in URLs provided by external sources such as backend servers. See `NetworkSecurityPolicy.isCleartextTrafficPermitted()` (/reference/android/security/NetworkSecurityPolicy#isCleartextTrafficPermitted()) for more details.

For example, you might want your app to ensure that connections to `secure.example.com` are always done over HTTPS to protect sensitive traffic from hostile networks.

The excerpt below shows how to opt out of cleartext in `res/xml/network_security_config.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config cleartextTrafficPermitted="false">
        <domain includeSubdomains="true">secure.example.com</domain>
    </domain-config>
</network-security-config>
```

Pin certificates

Normally, an app trusts all pre-installed CAs. If any of these CAs were to issue a fraudulent certificate, the app would be at risk from an on-path attacker. Some apps choose to limit the set of certificates they accept by either limiting the set of CAs they trust or by certificate pinning.

Certificate pinning is done by providing a set of certificates by hash of the public key (`SubjectPublicKeyInfo` of the X.509 certificate). A certificate chain is then valid only if the certificate chain contains at least one of the pinned public keys.

Note that, when using certificate pinning, you should always include a backup key so that if you are forced to switch to new keys or change CAs (when pinning to a CA certificate or an intermediate of that CA), your app's connectivity is unaffected. Otherwise, you must push out an update to the app to restore connectivity.

Additionally, it is possible to set an expiration time for pins after which pinning is not performed. This helps prevent connectivity issues in apps which have not been updated. However, setting an expiration time on pins may enable attackers to bypass your pinned certificates.

The excerpt below shows how to pin certificates in `res/xml/network_security_config.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config>
        <domain includeSubdomains="true">example.com</domain>
        <pin-set expiration="2018-01-01">
```

```
            <pin digest="SHA-256">7HIpactkIAq2Y49orFOOQKurWxmmSFZhBCoQYcRhJ3Y=</
            <!-- backup pin -->
            <pin digest="SHA-256">fwza0LRMXouZHRC8Ei+4PyuldPDcf3UKgO/04cDM1oE=</
        </pin-set>
    </domain-config>
</network-security-config>
```

Configuration inheritance behavior

Values not set in a specific configuration are inherited. This behavior allows more complex configurations while keeping the configuration file readable.

For example, values not set in a `domain-config` are taken from the parent `domain-config`, if nested, or from the `base-config`, if not. Values not set in the `base-config` use the platform default values.

For example, consider a case where all connections to subdomains of `example.com` must use a custom set of CAs. Additionally, cleartext traffic to these domains is permitted *except* when connecting to `secure.example.com`. By nesting the configuration for `secure.example.com` inside the configuration for `example.com`, the `trust-anchors` does not need to be duplicated.

The excerpt below shows how this nesting would look in `res/xml/network_security_config.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config>
        <domain includeSubdomains="true">example.com</domain>
        <trust-anchors>
            <certificates src="@raw/my_ca"/>
        </trust-anchors>
        <domain-config cleartextTrafficPermitted="false">
            <domain includeSubdomains="true">secure.example.com</domain>
        </domain-config>
    </domain-config>
</network-security-config>
```

Configuration file format

The Network Security Configuration feature uses an XML file format. The overall structure of the file is shown in the following code sample:

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <base-config>
        <trust-anchors>
            <certificates src="..."/>
            ...
        </trust-anchors>
    </base-config>

    <domain-config>
        <domain>android.com</domain>
        ...
        <trust-anchors>
            <certificates src="..."/>
            ...
        </trust-anchors>
        <pin-set>
            <pin digest="...">...</pin>
            ...
        </pin-set>
    </domain-config>
    ...
    <debug-overrides>
        <trust-anchors>
            <certificates src="..."/>
            ...
        </trust-anchors>
    </debug-overrides>
</network-security-config>
```

The following sections describe the syntax and other details of the file format.

<network-security-config>

can contain:

0 or 1 of **`<base-config>`** (#base-config)
Any number of **`<domain-config>`** (#domain-config)
0 or 1 of **`<debug-overrides>`** (#debug-overrides)

<base-config>

syntax:

```
<base-config cleartextTrafficPermitted (#CleartextTrafficPermitted)=["true" | "false"]>
    ...
</base-config>
```

can contain:

`<trust-anchors>` (#trust-anchors)

description:

The default configuration used by all connections whose destination is not covered by a **`domain-config`** (#domain-config).

Any values that are not set use the platform default values.

The default configuration for apps targeting Android 9 (API level 28) and higher is as follows:

```
<base-config cleartextTrafficPermitted="false">
    <trust-anchors>
        <certificates src="system" />
    </trust-anchors>
</base-config>
```

The default configuration for apps targeting Android 7.0 (API level 24) to Android 8.1 (API level 27) is as follows:

```
<base-config cleartextTrafficPermitted="true">
    <trust-anchors>
        <certificates src="system" />
    </trust-anchors>
</base-config>
```

The default configuration for apps targeting Android 6.0 (API level 23) and lower is as follows:

```
<base-config cleartextTrafficPermitted="true">
    <trust-anchors>
        <certificates src="system" />
        <certificates src="user" />
    </trust-anchors>
</base-config>
```

<domain-config>

syntax:

```
<domain-config cleartextTrafficPermitted (#CleartextTrafficPermitted)=["true" | "f
    ...
</domain-config>
```

can contain:

1 or more **<domain>** (#domain)
0 or 1 **<trust-anchors>** (#trust-anchors)
0 or 1 **<pin-set>** (#pin-set)
Any number of nested `<domain-config>`

description:

Configuration used for connections to specific destinations, as defined by the `domain` elements.

Note that if multiple `domain-config` elements cover a destination, the configuration with the most specific (longest) matching domain rule is used.

<domain>

syntax:

```
<domain includeSubdomains=["true" | "false"]>example.com</domain>
```

attributes:

`includeSubdomains`

If `"true"`, then this domain rule matches the domain and all subdomains, including subdomains of subdomains. Otherwise, the rule only applies to exact matches.

<debug-overrides>

syntax:

```
<debug-overrides>
    ...
</debug-overrides>
```

can contain:

0 or 1 `<trust-anchors>` (#trust-anchors)

description:

Overrides to be applied when android:debuggable (/guide/topics/manifest/application-element#debug) is `"true"`, which is normally the case for non-release builds generated by IDEs and build tools. Trust anchors specified in `debug-overrides` are added to all other configurations, and certificate pinning is not performed when the server's certificate chain uses one of these debug-only trust anchors. If android:debuggable (/guide/topics/manifest/application-element#debug) is `"false"`, then this section is completely ignored.

<trust-anchors>

syntax:

```
<trust-anchors>
...
</trust-anchors>
```

can contain:

Any number of `<certificates>` (#certificates)

description:

Set of trust anchors for secure connections.

<certificates>

syntax:

```
<certificates src=["system" | "user" | "raw resource"]
              overridePins=["true" | "false"] />
```

description:

Set of X.509 certificates for `trust-anchors` elements.

attributes:

`src`

The source of CA certificates. Each certificate can be one of the following:

- a raw resource ID pointing to a file containing X.509 certificates. Certificates must be encoded in DER or PEM format. In the case of PEM certificates, the file *must not* contain extra non-PEM data such as comments.
- `"system"` for the pre-installed system CA certificates
- `"user"` for user-added CA certificates

`overridePins`

Specifies if the CAs from this source bypass certificate pinning. If `"true"`, then pinning is not performed on certificate chains which are signed by one of the CAs from this source. This can be useful for debugging CAs or for testing man-in-the-middle attacks on your app's secure traffic.

Default is `"false"` unless specified in a `debug-overrides` element, in which case the default is `"true"`.

<pin-set>

syntax:

```
<pin-set expiration="date">
...
</pin-set>
```

can contain:

Any number of `<pin>` (#pin)

description:

A set of public key pins. For a secure connection to be trusted, one of the public keys in the chain of trust must be in the set of pins. See `<pin>` (#pin) for the format of pins.

attributes:

`expiration`

The date, in `yyyy-MM-dd` format, on which the pins expire, thus disabling pinning. If the attribute is not set, then the pins do not expire.

Expiration helps prevent connectivity issues in apps which do not get updates to their pin set, such as when the user disables app updates.

<pin>

syntax:

```
<pin digest=["SHA-256"]>base64 encoded digest of X.509
    SubjectPublicKeyInfo (SPKI)</pin>
```

attributes:

`digest`

The digest algorithm used to generate the pin. Currently, only `"SHA-256"` is supported.

Additional resources

For more information about Network Security Configuration, consult the following resources.

Codelabs

- Android Network Security Configuration (https://developer.android.com/codelabs/android-network-security-config)