Full Report for Identified Weaknesses

Function Call Flow for Weakness #1

When the app tries to add a Wi-Fi configuration: `addOrUpdateNetwork(config)` will be invoked.

```
public int addNetwork(WifiConfiguration config) {
        if (config == null) {
            return -1;
        }
        config.networkId = -1;
        return addOrUpdateNetwork(config);
    }
```

From this method the object `config` is generated as below:

```
public class WifiConfiguration implements Parcelable {
   public WifiConfiguration() {
   ...

   public static final @android.annotation.NonNull Creator<WifiConfiguration>
        CREATOR = new Creator<WifiConfiguration>() {
        ...
        config.enterpriseConfig = in.readParcelable(null);
        ...}
}
```

Furthermore, the function `setCaCertificate` can be set as null when creating the `WifiConfiguration` , which will allow the certificate to be "unspecified" or "Do not validate".

```
public class WifiEnterpriseConfig implements Parcelable {
    ...
    public WifiConfiguration() {...
    enterpriseConfig = new WifiEnterpriseConfig();
    ...
    }
    ...
    public static final @android.annotation.NonNull Creator<WifiEnterpriseConfig>
        CREATOR = new Creator<WifiEnterpriseConfig>() {...}
    ...
    public void setCaCertificate(@Nullable X509Certificate cert) {...}
```

```
    ...

}
```

Function Call Flow for Weakness #2

App will call methods in WifiManager to configure the wifi. To add a suggested network, `addNetworkSuggestions` is used.

```
//WifiManager.java
@RequiresPermission(android.Manifest.permission.CHANGE_WIFI_STATE)
    public @NetworkSuggestionsStatusCode int addNetworkSuggestions(
            @NonNull List<WifiNetworkSuggestion> networkSuggestions) {
        try {
            return mService.addNetworkSuggestions(
                    networkSuggestions, mContext.getOpPackageName(), mContext.getAttri
butionTag());
        } catch (RemoteException e) {
            throw e.rethrowFromSystemServer();
        }
    }
```

From this method, `mService.addNetworkSuggestions` from WifiServiceImpl.java is called.

```
//WifiServiceImpl.java
@Override
    public int addNetworkSuggestions(
            List<WifiNetworkSuggestion> networkSuggestions, String callingPackageName,
            String callingFeatureId) {
        if (enforceChangePermission(callingPackageName) != MODE_ALLOWED) {
            return WifiManager.STATUS_NETWORK_SUGGESTIONS_ERROR_APP_DISALLOWED;
        }
        if (mVerboseLoggingEnabled) {
            mLog.info("addNetworkSuggestions uid=%").c(Binder.getCallingUid()).flush
();
        }
        int callingUid = Binder.getCallingUid();

        int success = mWifiThreadRunner.call(() -> mWifiNetworkSuggestionsManager.add(
                networkSuggestions, callingUid, callingPackageName, callingFeatureId),
                WifiManager.STATUS_NETWORK_SUGGESTIONS_ERROR_INTERNAL);
        if (success != WifiManager.STATUS_NETWORK_SUGGESTIONS_SUCCESS) {
            Log.e(TAG, "Failed to add network suggestions");
        }
        return success;
    }
```

Then the add() function in WifiNetworkSuggestionsManager.java is called to add the suggested configuration. When there is already a configuration in

WifiConfigManager, `updateWifiConfigInWcmIfPresent` is called to update the configuration.

```
//WifiNetworkSuggestionsManager.java
/**
Add the provided list of network suggestions from the corresponding app's active list.
**/
public @WifiManager.NetworkSuggestionsStatusCode int add(
            List<WifiNetworkSuggestion> networkSuggestions, int uid, String packageNam
e,
            @Nullable String featureId) {
...
// If we have a config in WifiConfigManager for this suggestion, update
                // WifiConfigManager with the latest WifiConfig.
                // Note: Similar logic is present in PasspointManager for passpoint ne
tworks.
                updateWifiConfigInWcmIfPresent(
                        ewns.createInternalWifiConfiguration(), uid, packageName);
                addToScanResultMatchInfoMap(ewns);
...
}
```

```
//WifiNetworkSuggestionsManager.java
private void updateWifiConfigInWcmIfPresent(
            WifiConfiguration newConfig, int uid, String packageName) {
        WifiConfiguration configInWcm =
                mWifiConfigManager.getConfiguredNetwork(newConfig.getKey());
        if (configInWcm == null) return;
        // !suggestion
        if (!configInWcm.fromWifiNetworkSuggestion) return;
        // is suggestion from same app.
        if (configInWcm.creatorUid != uid
                || !TextUtils.equals(configInWcm.creatorName, packageName)) {
            return;
        }
        NetworkUpdateResult result = mWifiConfigManager.addOrUpdateNetwork(
                newConfig, uid, packageName);
        if (!result.isSuccess()) {
            Log.e(TAG, "Failed to update config in WifiConfigManager");
        } else {
            if (mVerboseLoggingEnabled) {
                Log.v(TAG, "Updated config in WifiConfigManager");
            }
        }
    }
```

In `updateWifiConfigInWcmIfPresent` method, `addOrUpdateNetwork` from WifiConfigManager is called.

```
//WifiConfigManager.java
public NetworkUpdateResult addOrUpdateNetwork(WifiConfiguration config, int uid) {
        return addOrUpdateNetwork(config, uid, null);
    }


private NetworkUpdateResult addOrUpdateNetworkInternal(WifiConfiguration config, int u
id,
                                                       @Nullable String packageNam
e) {
...
// Update the keys for saved enterprise networks. For Passpoint, the certificates
// and keys are installed at the time the provider is installed. For suggestion enterp
rise
        // network the certificates and keys are installed at the time the suggestion
 is added
        if (!config.isPasspoint() && !config.fromWifiNetworkSuggestion && config.isEnt
erprise()) {
            if (!(mWifiKeyStore.updateNetworkKeys(newInternalConfig, existingInternalC
onfig))) {
                return new NetworkUpdateResult(WifiConfiguration.INVALID_NETWORK_ID);
            }
        }
...
```

In the private method, `mWifiKeyStore.updateNetworkKeys` is called to update the keys for the wifi configuration.

```
public boolean updateNetworkKeys(WifiConfiguration config, WifiConfiguration existingC
onfig) {
        Preconditions.checkNotNull(mKeyStore);
        Preconditions.checkNotNull(config.enterpriseConfig);
        WifiEnterpriseConfig enterpriseConfig = config.enterpriseConfig;
        /* config passed may include only fields being updated.
         * In order to generate the key id, fetch uninitialized
         * fields from the currently tracked configuration
         */
        String keyId = config.getKeyIdForCredentials(existingConfig);
...
        try {
            if (!installKeys(existingEnterpriseConfig, enterpriseConfig, existingKeyI
d, keyId)) {
                Log.e(TAG, config.SSID + ": failed to install keys");
                return false;
            }
       }
...
}
```

In this method, the `getKeyIdForCredentials()` (method is mentioned in the patch) is called to derive the handle for the certificate (KeyID here is the reference to the wifi

configuration). What the patch does is to separate the already saved configuration from the suggested configuration. In the later section of the method, the `installKeys` method is called to update the X509 certificate in the wifi configurations. In this method, new certificates are added and old ones are removed.

The key insight for the patch is that the key ID for saved and suggested configuration is different such that the key update method will not confuse.

```
private boolean installKeys(WifiEnterpriseConfig existingConfig, WifiEnterpriseConfig
 config,
            String existingAlias, String alias) {
        Preconditions.checkNotNull(mKeyStore);
        Certificate[] clientCertificateChain = config.getClientCertificateChain();
        if (!ArrayUtils.isEmpty(clientCertificateChain)) {
            if (!putUserPrivKeyAndCertsInKeyStore(alias, config.getClientPrivateKey(),
                    clientCertificateChain)) {
                return false;
            }
        }
        X509Certificate[] caCertificates = config.getCaCertificates();
        Set<String> oldCaCertificatesToRemove = new ArraySet<>();
        if (existingConfig != null && existingConfig.getCaCertificateAliases() != nul
l) {
            oldCaCertificatesToRemove.addAll(
                    Arrays.asList(existingConfig.getCaCertificateAliases()));
        }
        List<String> caCertificateAliases = null;
        if (caCertificates != null) {
            caCertificateAliases = new ArrayList<>();
            for (int i = 0; i < caCertificates.length; i++) {
                String caAlias = String.format("%s_%d", alias, i);

                oldCaCertificatesToRemove.remove(caAlias);
                if (!putCaCertInKeyStore(caAlias, caCertificates[i])) {
                    // cleanup everything on failure.
                    removeEntryFromKeyStore(alias);
                    for (String addedAlias : caCertificateAliases) {
                        removeEntryFromKeyStore(addedAlias);
                    }
                    return false;
                }
                caCertificateAliases.add(caAlias);
            }
        }
        // If alias changed, remove the old one.
        if (!alias.equals(existingAlias)) {
            // Remove old private keys.
            removeEntryFromKeyStore(existingAlias);
        }
        // Remove any old CA certs.
        for (String oldAlias : oldCaCertificatesToRemove) {
            removeEntryFromKeyStore(oldAlias);
        }
        // Set alias names
```

```
        if (config.getClientCertificate() != null) {
            config.setClientCertificateAlias(alias);
            config.resetClientKeyEntry();
        }

        if (caCertificates != null) {
            config.setCaCertificateAliases(
                    caCertificateAliases.toArray(new String[caCertificateAliases.size
()]));
            config.resetCaCertificate();
        }
        return true;
    }
```