Full Report for Identified Weaknesses

In this report, for each identified weaknesses, we first present a summarized graph for the function call flow, followed by a detailed explanation.

1. Function Call Flow for Weakness #1

1.1 Summarized Call Flow

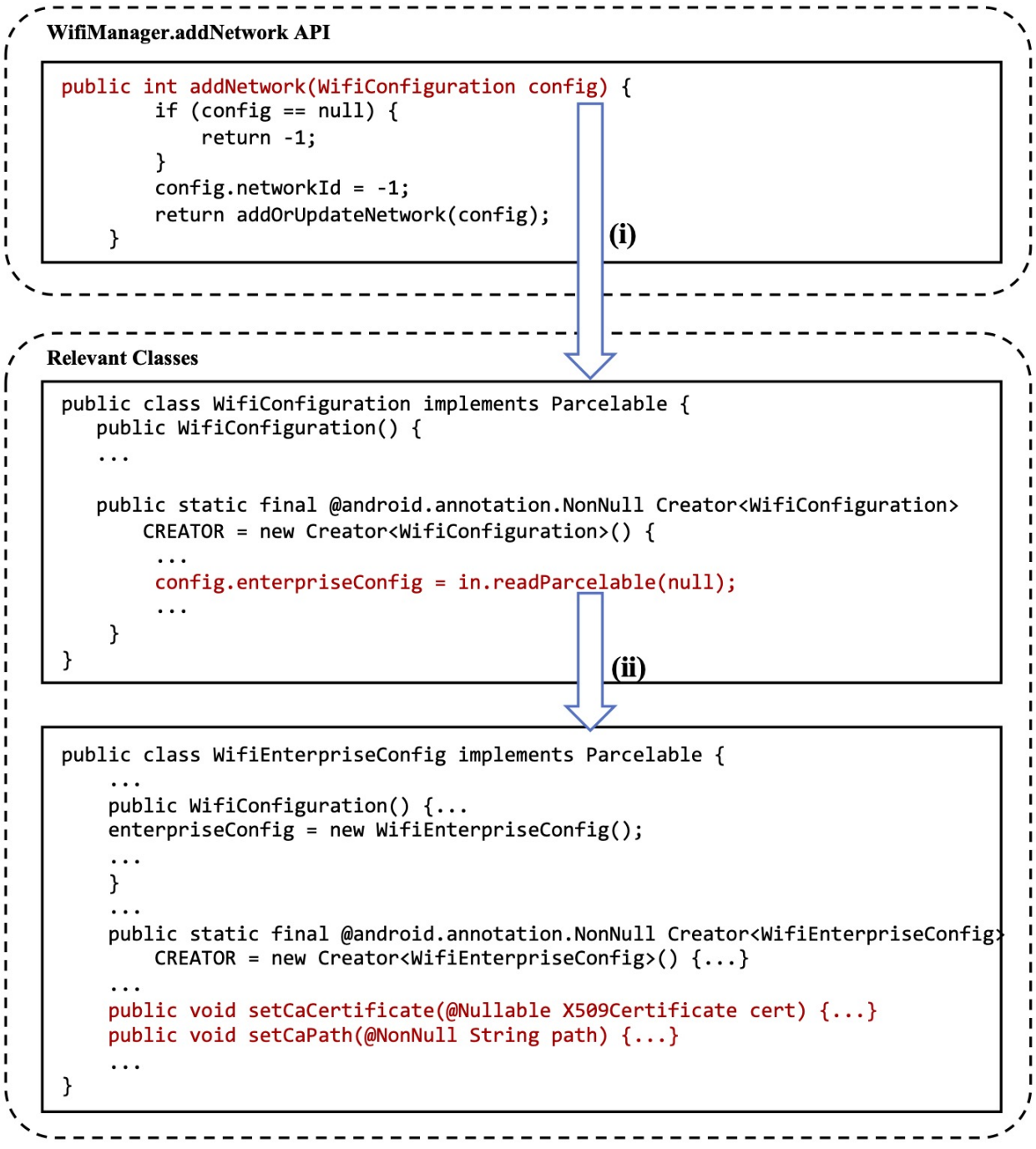

Figure 1. Summarized Function Call for Weakness #1

1.2 Detailed Function Call Flow

When the app tries to add a Wi-Fi configuration: `addOrUpdateNetwork(config)` will be invoked, as shown in Codeblock 1.

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

From this `addOrUpdateNetwork(config)` method the object `config` is generated by `WifiConfiguration` class the as shown in Codeblock 2.

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

   public static final @android.annotation.NonNull Creator<WifiConfiguration>
        CREATOR = new Creator<WifiConfiguration>() {
```

```
        ...
        config.enterpriseConfig = in.readParcelable(null);
        ...}
}
```

As shown in Codeblock 3, the function `setCaCertificate` can be set as null when creating the configuration object `WifiConfiguration` within the `WifiEnterpriseConfig` class , 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) {...}
    ...

}
```

2. Function Call Flow for Weakness #2

2.1 Summarized Call Flow

Figure 2. Summarized Function Call for Weakness #2

2.2 Detailed Function Call Flow

The Wi-Fi configuration App (BYOD app in the paper) will call methods in WifiManager to configure the wifi. To add a suggested network, `addNetworkSuggestions` is used, as shown in Codeblock 4.

```
//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.getAttributionTag());
        } catch (RemoteException e) {
            throw e.rethrowFromSystemServer();
        }
    }
```

From this `addNetworkSuggestions` method, `mService.addNetworkSuggestions` from WifiServiceImpl.java is called, as shown in Codeblock 4. Then the `add()` function in WifiNetworkSuggestionsManager.java is called to add the suggested configuration, as shown in Codeblock 5.

```
//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, callingPackageNam
e, callingFeatureId),
                WifiManager.STATUS_NETWORK_SUGGESTIONS_ERROR_INTERNAL);
        if (success != WifiManager.STATUS_NETWORK_SUGGESTIONS_SUCCESS) {
            Log.e(TAG, "Failed to add network suggestions");
        }
        return success;
    }
```

The function `add()` in WifiNetworkSuggestionsManager.java is called to add the suggested configuration. If there is already a configuration in WifiConfigManager, `updateWifiConfigInWcmIfPresent` is called to update the configuration, as shown in Codeblock 6.

```
//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 packageName,
            @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 networks.
                updateWifiConfigInWcmIfPresent(
                        ewns.createInternalWifiConfiguration(), uid, packageName);
                addToScanResultMatchInfoMap(ewns);
...
}
```

In `updateWifiConfigInWcmIfPresent` method, `addOrUpdateNetwork` from WifiConfigManager is called, as shown in Codeblock 7.

```
//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");
            }
        }
    }
```

Furthermore, in the private method `addOrUpdateNetworkInternal`, `mWifiKeyStore.updateNetworkKeys` is called to update the keys for the wifi configuration as shown in Codeblock 8.

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

private NetworkUpdateResult addOrUpdateNetworkInternal(WifiConfiguration config, int uid,@Nullable String packageName) {
...
// Update the keys for saved enterprise networks. For Passpoint, the certificates
// and keys are installed at the time the provider is installed. For suggestion enterprise
        // network the certificates and keys are installed at the time the suggestion is added
        if (!config.isPasspoint() && !config.fromWifiNetworkSuggestion && config.isEnterprise()) {
            if (!(mWifiKeyStore.updateNetworkKeys(newInternalConfig, existingInternalConfig))) {
                return new NetworkUpdateResult(WifiConfiguration.INVALID_NETWORK_ID);
            }
        }
...
```

In this private method `updateNetworkKeys`, 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), as shown in Codeblock 8. The patch separates 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.

```
//WifiKeyStore.java
public boolean updateNetworkKeys(WifiConfiguration config, WifiConfiguration existingConfig) {
        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, existingKeyId, keyId)) {
                Log.e(TAG, config.SSID + ": failed to install keys");
                return false;
            }
       }
...
}
```

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, as shown in Codeblock 10 where `installKeys` is called.

```
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() != null) {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;
    }
```