Android Wifi Direct Analysis

보고서 및 논문 윤리 서약

1. 나는 보고서 및 논문의 내용을 조작하지 않겠습니다.
2. 나는 다른 사람의 보고서 및 논문의 내용을 내 것처럼 무단으로 복사하지 않겠습니다.
3. 나는 다른 사람의 보고서 및 논문의 내용을 참고하거나 인용할 시 참고 및 인용 형식을 갖추고 출처를 반드시 밝히겠습니다.
4. 나는 보고서 및 논문을 대신하여 작성하도록 청탁하지도 청탁받지도 않겠습니다.

나는 보고서 및 논문 작성 시 위법 행위를 하지 않고, 명지인으로서 또한 공학인으로서 나의 양심과 명예를 지킬 것을 약속합니다.

보고서명 : Android Wifi Direct Analysis

명지대학교 보안경영공학과 김진성
nadas9029@gmail.com

1. 목적

Cloudlet의 보안을 점검하기 위해 Android의 WifiP2P 구현 소스를 분석하고 어떤 보안기술이 적용되어 있는지 확인한다.

2. 배경

안드로이드 스마트 폰에서 Wifi Direct를 연결할 때 보안 연결에 대한 설정 부분이 없다. 안드로이드 소스코드를 분석해서 어떤 보안을 사용하는지 확인해야 한다. 또한 사용한다면 키 생성은 어떻게 하는지 확인이 필요하다. 안드로이드 소스코드 버전은 4.4.4 Kitkat을 기반으로 한다.

3. 코드분석 정리

WifiP2pManager.java

```
public Channel initialize(Context srcContext, Looper srcLooper, ChannelListener
listener) {
      Messenger messenger = getMessenger();
      if (messenger == null) return null;

      Channel c = new Channel(srcContext, srcLooper, listener);
      if (c.mAsyncChannel.connectSync(srcContext, c.mHandler, messenger)
                  == AsyncChannel.STATUS_SUCCESSFUL) {
            return c;
      } else {
            return null;
      }
}
```

◎Wifi Direct를 사용하기 위해서 처음으로 호출되어야 하는 함수

◎Wifi Direct에서 반드시 필요한 Channel을 return

```
public void discoverPeers(Channel c, ActionListener listener) {
            checkChannel(c);
            c.mAsyncChannel.sendMessage(DISCOVER_PEERS, 0, c.putListener(listener));
      }
```

◎이용가능한 Wi-Fi peers를 찾는다.

```
public void connect(Channel c, WifiP2pConfig config, ActionListener listener) {
            checkChannel(c);
            checkP2pConfig(config);
            c.mAsyncChannel.sendMessage(CONNECT, 0, c.putListener(listener), config);
      }
```

◎p2p 연결을 시작한다.

◎여기서 인자로 사용하는 WifiP2pConfig 클래스에서 WPS에 대한 설정을 하는 것을 확인했다. 그렇기 때문에 WifiP2pConfig 클래스를 분석할 필요가 있다.

WifiP2pConfig.java

변수

```
String deviceAddress
WpsInfo wps
static final int MAX_GROUP_OWNER_INTENT = 15
static final int MIN_GROUP_OWNER_INTENT = 0
int groupOwnerIntent
int netId
```

◎WPS의 정보를 가지고 있는 WpsInfo를 가지고 있다.

◎Group Owner를 결정하기 위한 GROUP_OWNER_INTENT의 최대 최소값을 가지고 있고 GROUP_OWNER_INETEN가 높은 디바이스가 Group Owner가 된다.

함수

```
public WifiP2pConfig() {
    //set defaults
    wps = new WpsInfo();
    wps.setup = WpsInfo.PBC;
}
```

◎기본 생성자

◎WpsInfo()를 만들어서 default 설정값인 PBC로 설정한다.

```
public WifiP2pConfig(String supplicantEvent) throws IllegalArgumentException {
    String[] tokens = supplicantEvent.split(" ");

    if (tokens.length < 2 || !tokens[0].equals("P2P-GO-NEG-REQUEST")) {
        throw new IllegalArgumentException("Malformed supplicant event");
    }

    deviceAddress = tokens[1];
    wps = new WpsInfo();

    if (tokens.length > 2) {
        String[] nameVal = tokens[2].split("=");
        int devPasswdId;
        try {
            devPasswdId = Integer.parseInt(nameVal[1]);
        } catch (NumberFormatException e) {
            devPasswdId = 0;
        }
        //Based on definitions in wps/wps_defs.h
        switch (devPasswdId) {
            //DEV_PW_USER_SPECIFIED = 0x0001,
            case 0x01:
                wps.setup = WpsInfo.DISPLAY;
                break;
            //DEV_PW_PUSHBUTTON = 0x0004,
            case 0x04:
                wps.setup = WpsInfo.PBC;
                break;
            //DEV_PW_REGISTRAR_SPECIFIED = 0x0005
            case 0x05:
                wps.setup = WpsInfo.KEYPAD;
                break;
            default:
                wps.setup = WpsInfo.PBC;
                break;
        }
    }
}
```

◎supplicantEvent 스트링을 넘겨받는 생성자

◎공백(“ ”)을 기준으로 split하여 P2P-GO-NEG-REQUEST이면 정상적인 호출로 확인한다.

◎정상적인 호출로 판단하면 deviceAddress를 설정하고 WpsInfo를 생성한다.

◎그리고 devPasswdId 부분을 분석하여 각자에 맞는 설정을 해준다.

◎default 설정은 PBC를 사용하는 것으로 한다.

◎DISPLAY 설정은 pin을 생성하여 화면에 보여줌

◎PBC 설정은 push button configuration 설정함

◎KEYPAD 설정은 디바이스를 통해서 PIN 값을 입력받음

◎위 내용을 살펴봤을 때 String supplicantEvent의 내용은 다음과 같은 형식으로 볼 수 있다. P2P-GO-NEG-REQUEST {deviceAddress} dev_passwd_id={CODE}

```
public WifiP2pConfig(WifiP2pConfig source) {
    if (source != null) {
        deviceAddress = source.deviceAddress;
        wps = new WpsInfo(source.wps);
        groupOwnerIntent = source.groupOwnerIntent;
        netId = source.netId;
    }
}
```

◎복사 생성자

◎deviceAddress, wpsInfo, groupOwnerIntent, netId를 복사한다.

WifiP2pService.java

class InactiveState

```
public boolean processMessage(Message message) {
        if (DBG) logd(getName() + message.toString());
        switch (message.what) {
            case WifiP2pManager.CONNECT:
                if (DBG) logd(getName() + " sending connect");
                WifiP2pConfig config = (WifiP2pConfig) message.obj;
                if (isConfigInvalid(config)) {
                    loge("Dropping connect requeset " + config);
                    replyToMessage(message, WifiP2pManager.CONNECT_FAILED);
                    break;
                }

                mAutonomousGroup = false;
                mWifiNative.p2pStopFind();
                if (reinvokePersistentGroup(config)) {
                    transitionTo(mGroupNegotiationState);
                } else {
                    transitionTo(mProvisionDiscoveryState);
                }
                mSavedPeerConfig = config;
                mPeers.updateStatus(mSavedPeerConfig.deviceAddress,
                                                    WifiP2pDevice.INVITED);
                sendPeersChangedBroadcast();
                replyToMessage(message,
                                        WifiP2pManager.CONNECT_SUCCEEDED);
                break;
```

◎WifiP2pManager.java의 connect를 통해서 보낸 message를 처리한다.

◎isConfigInvalid(config)는 deviceAddress에 대한 유효성 검사

◎reinvokePersistentGroup(config)는 netID에 대한 처리

class GroupCreatedState

```
case WifiP2pManager.CONNECT:
    WifiP2pConfig config = (WifiP2pConfig) message.obj;
    if (isConfigInvalid(config)) {
        loge("Dropping connect requeset " + config);
        replyToMessage(message, WifiP2pManager.CONNECT_FAILED);
        break;
    }
    logd("Inviting device : " + config.deviceAddress);
    mSavedPeerConfig = config;
    if (mWifiNative.p2pInvite(mGroup, config.deviceAddress)) {
        mPeers.updateStatus(config.deviceAddress, WifiP2pDevice.INVITED);
        sendPeersChangedBroadcast();
        replyToMessage(message, WifiP2pManager.CONNECT_SUCCEEDED);
    } else {
        eplyToMessage(message, WifiP2pManager.CONNECT_FAILED,
                                                    WifiP2pManager.ERROR);
    }
// TODO: figure out updating the status to declined when invitation is rejected
    break;
```

◎GroupCreatedState에서의 .CONNECT처리

◎이 CONNECT처리에서도 deviceAddress만을 처리한다.

```
case WifiP2pManager.START_WPS:
    WpsInfo wps = (WpsInfo) message.obj;
    if (wps == null) {
        replyToMessage(message, WifiP2pManager.START_WPS_FAILED);
        break;
    }
    boolean ret = true;
    if (wps.setup == WpsInfo.PBC) {
        ret = mWifiNative.startWpsPbc(mGroup.getInterface(), null);
    } else {
        if (wps.pin == null) {
            String pin = mWifiNative.startWpsPinDisplay(mGroup.getInterface());
            try {
                Integer.parseInt(pin);
                notifyInvitationSent(pin, "any");
            } catch (NumberFormatException ignore) {
                ret = false;
            }
        } else {
            ret = mWifiNative.startWpsPinKeypad(mGroup.getInterface(), wps.pin);
        }
     }
     replyToMessage(message, ret ? WifiP2pManager.START_WPS_SUCCEEDED :
                                    WifiP2pManager.START_WPS_FAILED);
     break;
```

◎GroupCreatedState에서의 START_WPS 처리부분

◎WpsInfo.PBC로 설정되어 있을 때는 mWifiNative.startWpsPbc()로 처리

◎pin이 null일 때는 Display로 처리하고 null이 아닐때는 Keypad로 처리한다.

◎문제 : 어디서 START_WPS를 보내는지 확인하지 못함

◎각 Native 함수에 대한 분석이 필요

4. 결론

처음 연결 시 WPS를 이용한다. connect 함수를 호출하면서 사용하는 WifiP2pConfig 객체를 통해서 WpsConfig를 설정하는 것을 확인했다. 하지만 WifiP2pService.java에서 WifiP2pManager.START_WPS를 어디에서 보내는 것인지 확인하지 못했다. 또한 이후 통신에서 사용할 WPA2를 위한 설정이 어디에서 이루어지는지 확인하지 못했다. 이 설정을 확인하기 위해서 CONNECT이후 수행된다고 나와 있는 몇몇 함수를 확인해봤지만 확인하지 못했다.

WifiP2pService.java에서 wpa_supplicant를 많이 볼 수 있는데 검색 결과 안드로이드에서는 wpa_supplicant를 이용해서 WPA2를 구현한 것 같다. 어떻게 키가 생성되고 통신하는지 좀 더 살펴볼 필요가 있다. wpa_supplicant의 자세한 내용은 w1.fi/wpa_supplicant/에서 확인할 수 있다.