# TODO

# Wifi

frameworks/opt/net/wifi/libwifi\_system/hostapd\_manager.cpp

关键字：

## libwifi\_system

### HostapdManager

**const int** kDefaultApChannel = 6;  
**const char** kHostapdServiceName[] = **"hostapd"**;  
**const char** kHostapdConfigFilePath[] = **"/data/misc/wifi/hostapd.conf"**;

#### StartHostapd

启动热点很简单。。。这也太随意了吧

Setprop **ctl.start hostapd**

同理关闭 Setprop **ctl.stop hostapd**

这里其实没有实现互斥啊

**bool** HostapdManager::StartHostapd() {  
 **if** (!SupplicantManager::EnsureEntropyFileExists()) {  
 LOG(WARNING) << **"Wi-Fi entropy file was not created"**;  
 }  
  
 **if** (property\_set(**"ctl.start"**, kHostapdServiceName) != 0) {  
 LOG(ERROR) << **"Failed to start SoftAP"**;  
 **return false**;  
 }  
  
 LOG(DEBUG) << **"SoftAP started successfully"**;  
 **return true**;  
}

#### WriteHostapdConfig

**bool** HostapdManager::WriteHostapdConfig(**const** string& config) {  
 **if** (!WriteStringToFile(config, kHostapdConfigFilePath,  
 S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IWGRP,  
 AID\_WIFI, AID\_WIFI)) {  
 **int** error = errno;  
 LOG(ERROR) << **"Cannot write hostapd config to \""** << kHostapdConfigFilePath << **"\": "** << strerror(error);  
 **struct** stat st;  
 **int** result = stat(kHostapdConfigFilePath, &st);  
 **if** (result == 0) {  
 LOG(ERROR) << **"hostapd config file uid: "**<< st.st\_uid << **", gid: "** << st.st\_gid  
 << **", mode: "** << st.st\_mode;  
 } **else** {  
 LOG(ERROR) << **"Error calling stat() on hostapd config file: "** << strerror(errno);  
 }  
 **return false**;  
 }  
 **return true**;  
}

#### CreateHostapdConfig

string HostapdManager::CreateHostapdConfig(  
 **const** string& interface\_name,  
 **const** vector<uint8\_t> ssid,  
 **bool** is\_hidden,  
 **int** channel,  
 EncryptionType encryption\_type,  
 **const** vector<uint8\_t> passphrase) {  
 string result;  
  
 **if** (channel < 0) {  
 channel = kDefaultApChannel;  
 }  
  
 **if** (ssid.size() > 32) {  
 LOG(ERROR) << **"SSIDs must be <= 32 bytes long"**;  
 **return** result;  
 }  
  
 stringstream ss;  
 ss << std::hex;  
 ss << std::setfill(**'0'**);  
 **for** (uint8\_t b : ssid) {  
 ss << std::setw(2) << **static\_cast**<**unsigned int**>(b);  
 }  
 **const** string ssid\_as\_string = ss.str();  
  
 string encryption\_config;  
 **if** (encryption\_type != EncryptionType::kOpen) {  
 string psk = GeneratePsk(ssid, passphrase);  
 **if** (psk.empty()) {  
 **return** result;  
 }  
 **if** (encryption\_type == EncryptionType::kWpa) {  
 encryption\_config = StringPrintf(**"wpa=3\n"  
 "wpa\_pairwise=TKIP CCMP\n"  
 "wpa\_psk=%s\n"**, psk.c\_str());  
 } **else if** (encryption\_type == EncryptionType::kWpa2) {  
 encryption\_config = StringPrintf(**"wpa=2\n"  
 "rsn\_pairwise=CCMP\n"  
 "wpa\_psk=%s\n"**, psk.c\_str());  
 } **else** {  
 **using** encryption\_t = std::underlying\_type<EncryptionType>::type;  
 LOG(ERROR) << **"Unknown encryption type ("** << **static\_cast**<encryption\_t>(encryption\_type)  
 << **")"**;  
 **return** result;  
 }  
 }  
  
 result = StringPrintf(  
 **"interface=%s\n"  
 "driver=nl80211\n"  
 "ctrl\_interface=/data/misc/wifi/hostapd/ctrl\n"** *// ssid2 signals to hostapd that the value is not a literal value  
 // for use as a SSID. In this case, we're giving it a hex string  
 // and hostapd needs to expect that.* **"ssid2=%s\n"  
 "channel=%d\n"  
 "ieee80211n=1\n"  
 "hw\_mode=%c\n"  
 "ignore\_broadcast\_ssid=%d\n"  
 "wowlan\_triggers=any\n"  
 "%s"**,  
 interface\_name.c\_str(),  
 ssid\_as\_string.c\_str(),  
 channel,  
 (channel <= 14) ? **'g'** : **'a'**,  
 (is\_hidden) ? 1 : 0,  
 encryption\_config.c\_str());  
 **return** result;  
}

## Service

### jni

### hotspot2

# 实战

## 监听设备接入

01-18 17:05:40.066 567-567/? I/wificond: New station ec:d0:9f:b0:df:97 associated with hotspot

01-18 17:05:39.048 582-582/? I/chatty: uid=1000(system) /system/bin/ATFWD-daemon identical 1 line

01-18 17:05:40.049 582-582/? I/ServiceManager: Waiting for service AtCmdFwd...

01-18 17:05:40.066 4530-4530/? I/hostapd: wlan0: STA ec:d0:9f:b0:df:97 IEEE 802.11: associated

wlan0: STA ec:d0:9f:b0:df:97 IEEE 802.11: associated

wlan0: AP-STA-CONNECTED ec:d0:9f:b0:df:97

通过日志来看，我们至少可以从wificond或者hostapd来寻找分析，由于开放的是热点，因此我们直接分析hostapd。

wificond

# REF