https://github.com/awedxz/hgame2018-

WEEK3

babyRSA

用 `openssl rsautl -decrypt -in flag.enc -inkey private.pem` 去解密会报错

查看错误信息大致可以猜测到是RSA填充的问题 百度一下就能找到oaep了

`openssl rsautl -decrypt -oaep -in flag.enc -inkey private.pem` 得到flag

NotHardRSA

5个素数用了10遍 肯定有重复用的了

设生成的素数 $p_1p_2p_3p_4p_5$ 那么可以得到以下式子

$$c1 \equiv m^e \pmod{(p_1 * p_2)} \Rightarrow m^e = (p_1 * p_2) * k_1 + c_1$$

$$c_2 \equiv m^e \pmod{(p_1 * p_3)} \Rightarrow m^e = (p_1 * p_3) * k_2 + c_2$$

$$c_3 \equiv m^e \pmod{(p_1 * p_4)} \Rightarrow m^e = (p_1 * p_4) * k_3 + c_3$$

一式减二式 一式减三式 移项可得

$$c_1 - c_2 = p_1 * (k_2 * p_3 - k_1 * p_2)$$

$$c_1 - c_3 = p_1 * (k_3 * p_4 - k_1 * p_2)$$

到此我们发现 $c_1 - c_2$ $c_1 - c_3$ 共用了一个因子 p_1 所以这题就是一道变化了个共享素数

接下来的过程就是常规的共享素数了 参考solve.py

CBC V0.1

就是单纯的 `padding oracle attack` 源代码参照index.php

网上解释挺详细的 在此不再赘述 脚本参考solve.py

再解释一下 这里的 `decrypt error` 就是填充错误的意思

CBC V0.2

虽然表面上看上去像CBC字节翻转和哈希长度扩展攻击的联合利用

但是实际上因为条件的限制 我们无法使用lea 因为我们不知道我们输入的密文的解密结果

其实漏洞就在代码的第15行

```
1   unpad = lambda s : s[0:-ord(s[-1])]
```

这里的并没有检测s 那么如果我们的s是下面这样的呢

```
1   s = "admin\x0a\x0a\x0a\x0a\x0a\x0a\x0a\x0a\x0a\x0a\xff"
```

这样经过unpad处理之后 `s == ''` 从而md5(s)的值被固定为 `d41d8cd98f00b204e9800998ecf8427e`

剩下就是字节翻转了 网上有大量文章 在此不再赘述 详情参考solve.py

正常的SQLi

.bak可以找到部分源代码 可以看到代码中输出flag被注释掉了 回显没了 乍一看可能不知道怎么做

但是回显没了 我们可以创造回显 网上搜搜可以找到一种注入方式--延时盲注

以下为sqlmap一把梭payload

```
1   python sqlmap.py -u "http://123.206.203.108:10010/normalSQLi/index.php" --cookie="name=" --
    tamper "base64encode" --random-agent --level 3 --technique "T" -v 3 -D users -T user -C
    "id,password,username" --dump
```

这里 `-v 3` 显示了所有注入的payload 不会的可以学习一下