谈HTTPS中间人攻击与证书校验（二）

上文说到HTTPS的三次握手:http://www.cnblogs.com/wh4am1/p/6616851.html

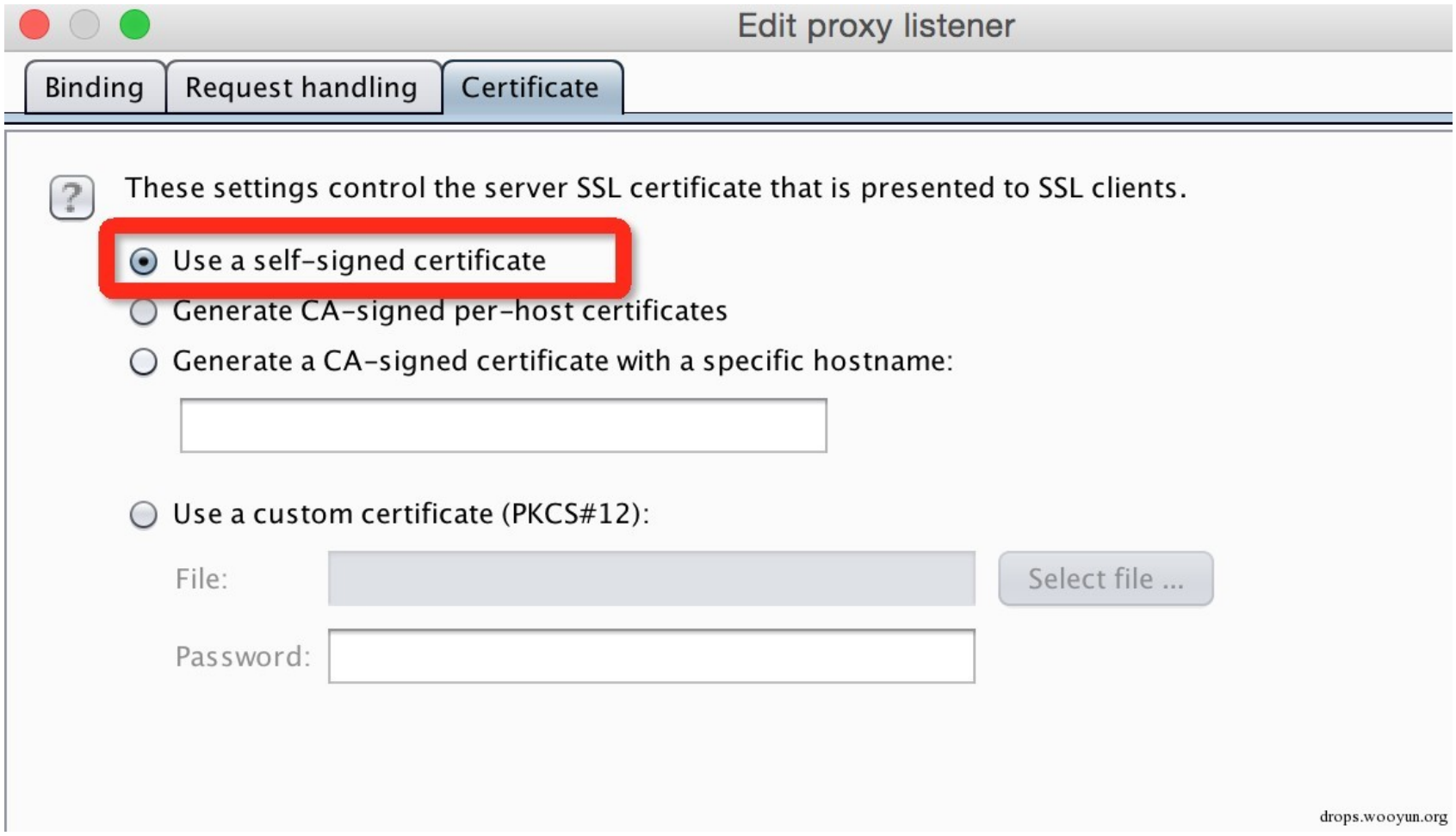
不懂的再回头去看看

三、中间人攻击

https握手过程的证书校验环节就是为了识别证书的有效性唯一性等等，所以严格意义上来说https下不存在中间人攻击，存在中间人攻击的前提条件是没有严格的对证书进行校验，或者人为的信任伪造证书，下面一起看下几种常见的https“中间人攻击”场景。

1.证书未校验

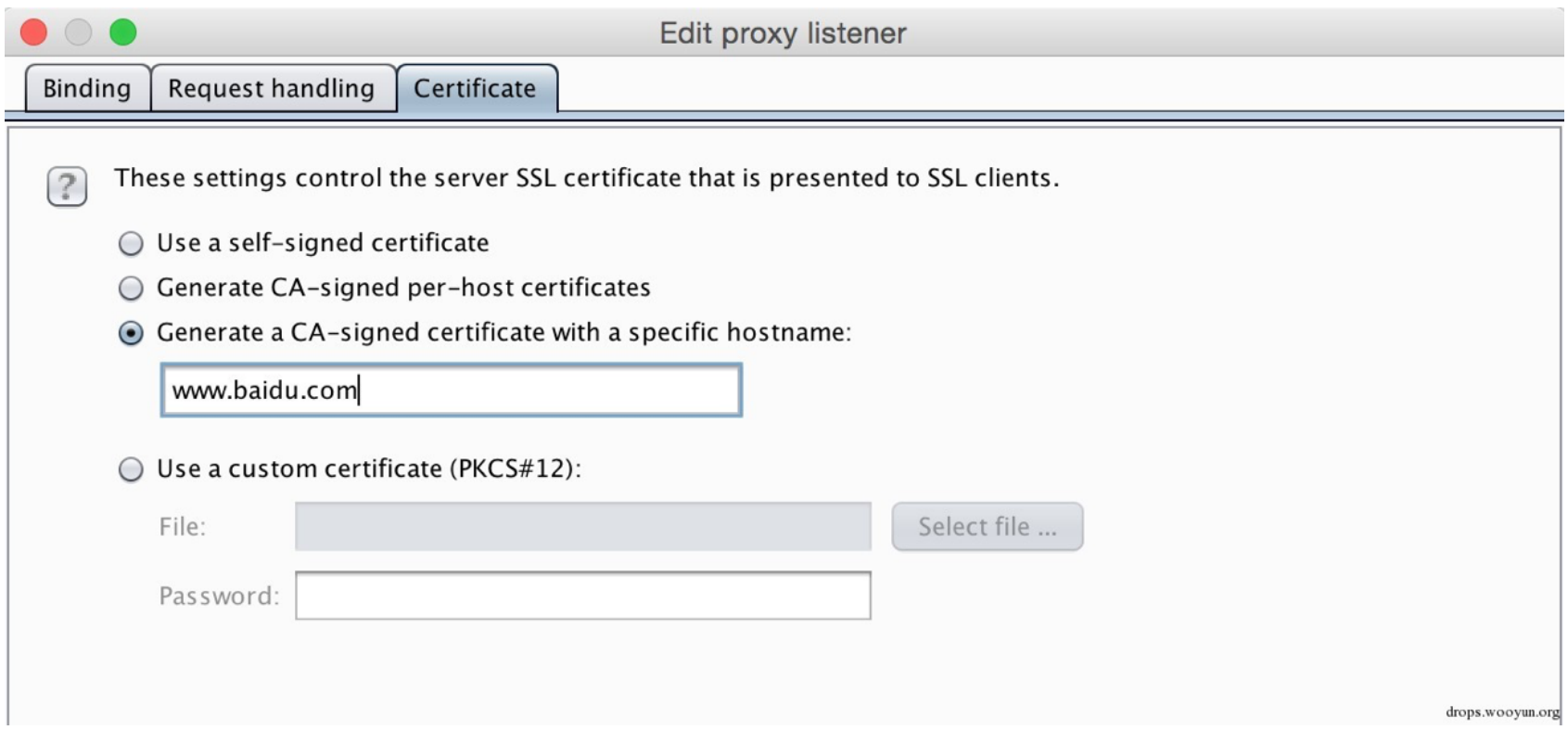
由于客户端没有做任何的证书校验，所以此时随意一张证书都可以进行中间人攻击，可以使用burp里的这个模块进行中间人攻击。



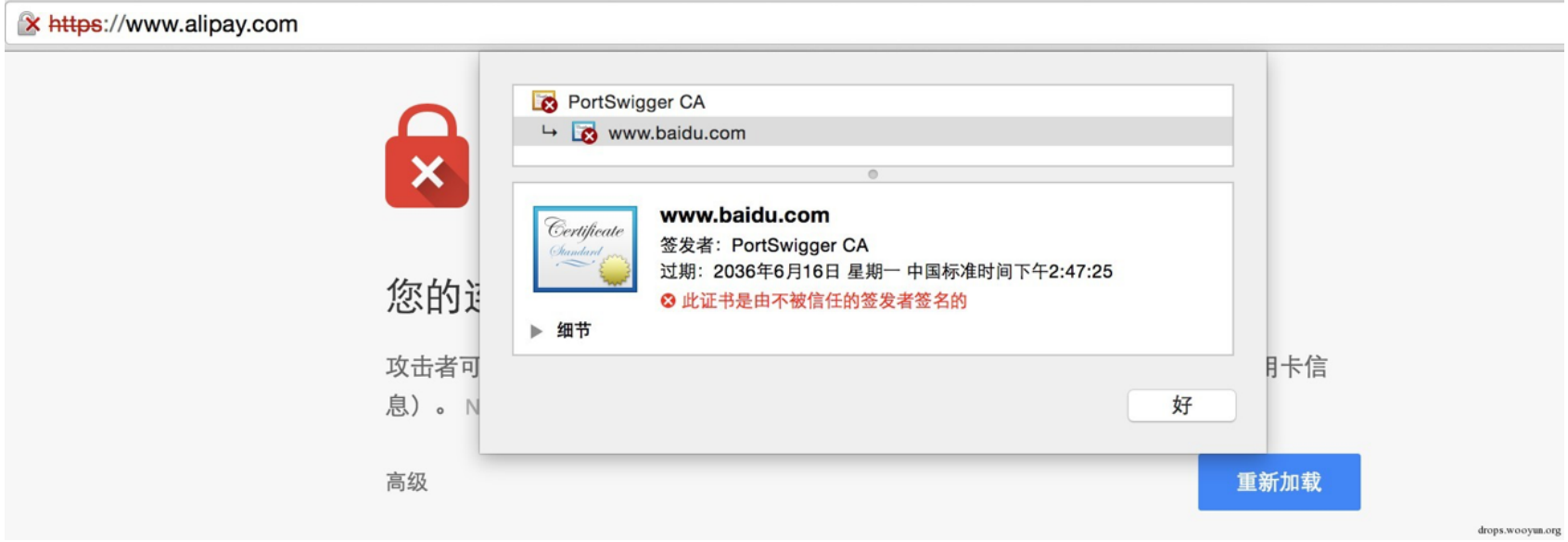
通过浏览器查看实际的https证书，是一个自签名的伪造证书。

2.部分校验

做了部分校验，例如在证书校验过程中只做了证书域名是否匹配的校验，可以使用burp的如下模块生成任意域名的伪造证书进行中间人攻击。

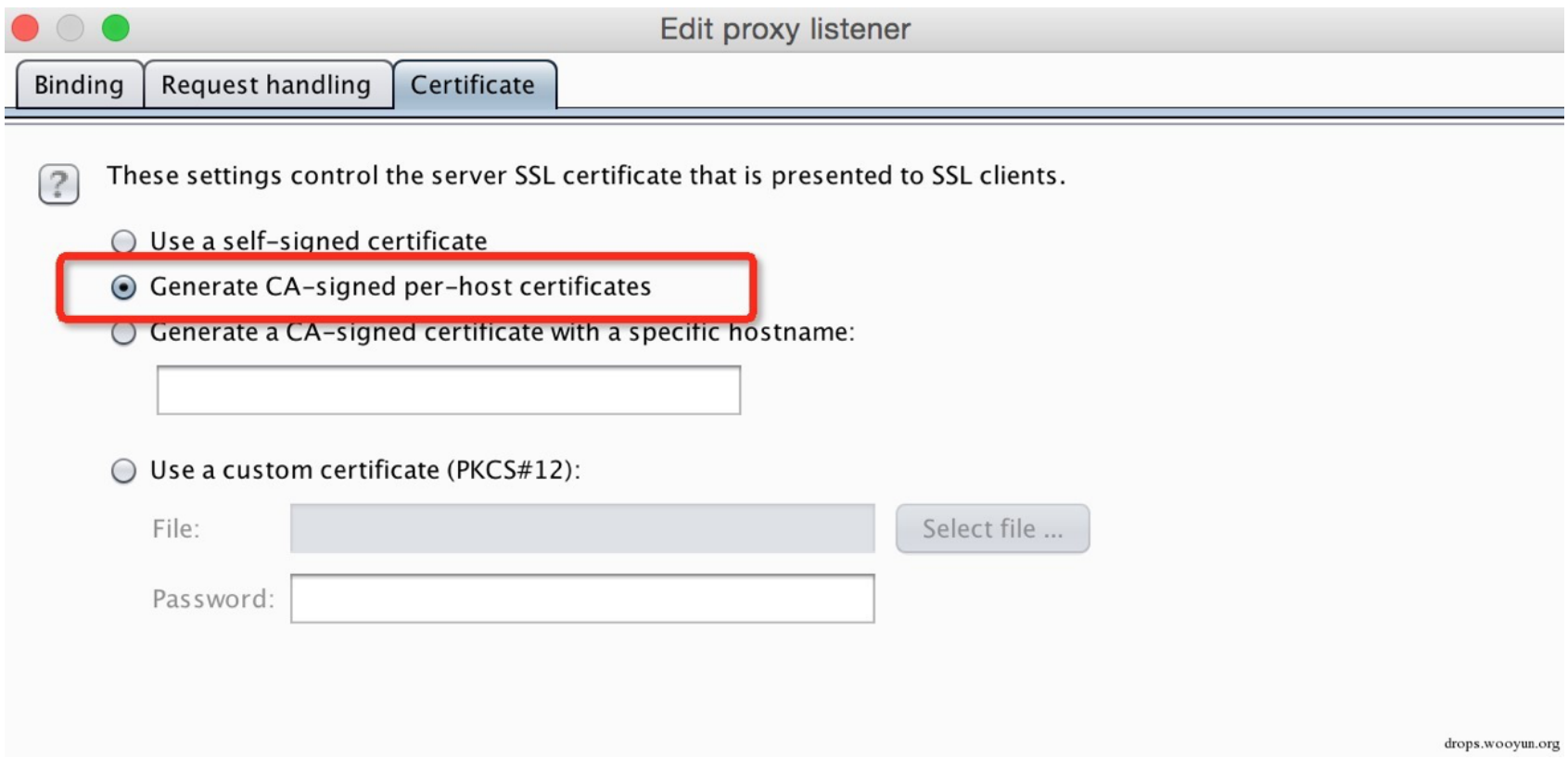


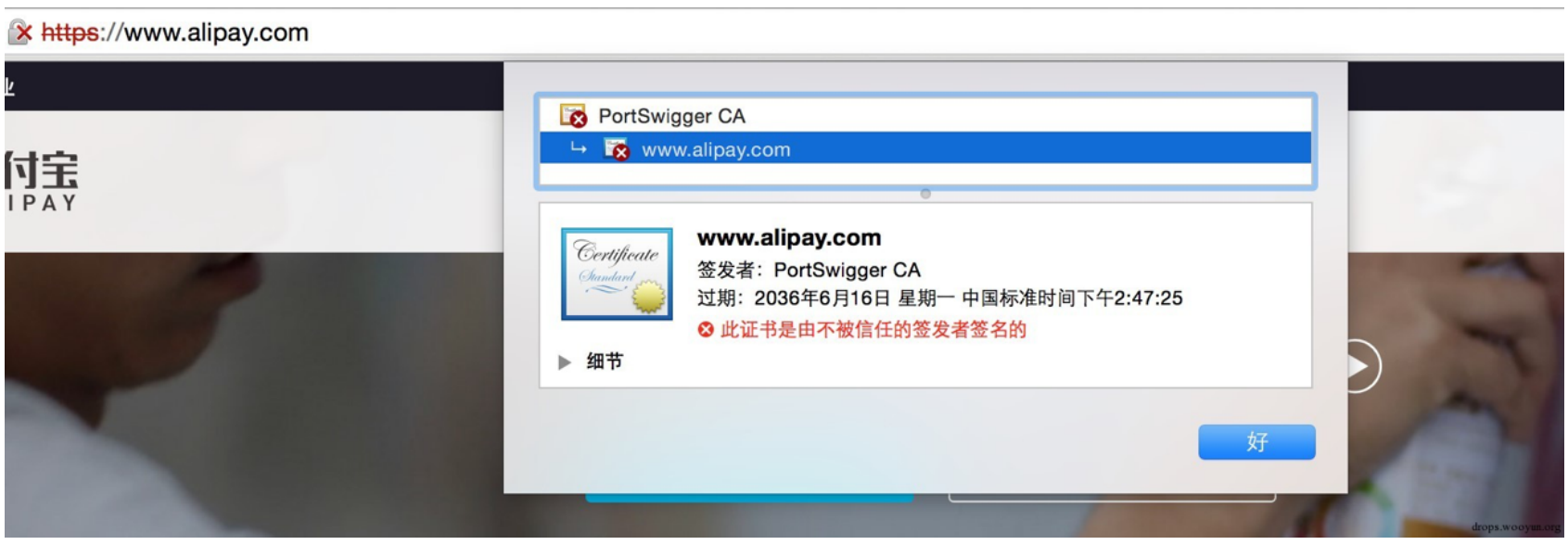
实际生成的证书效果，如果只做了域名、证书是否过期等校验可轻松进行中间人攻击(由于chrome是做了证书校验的所以会提示证书不可信任)。



3.证书链校验

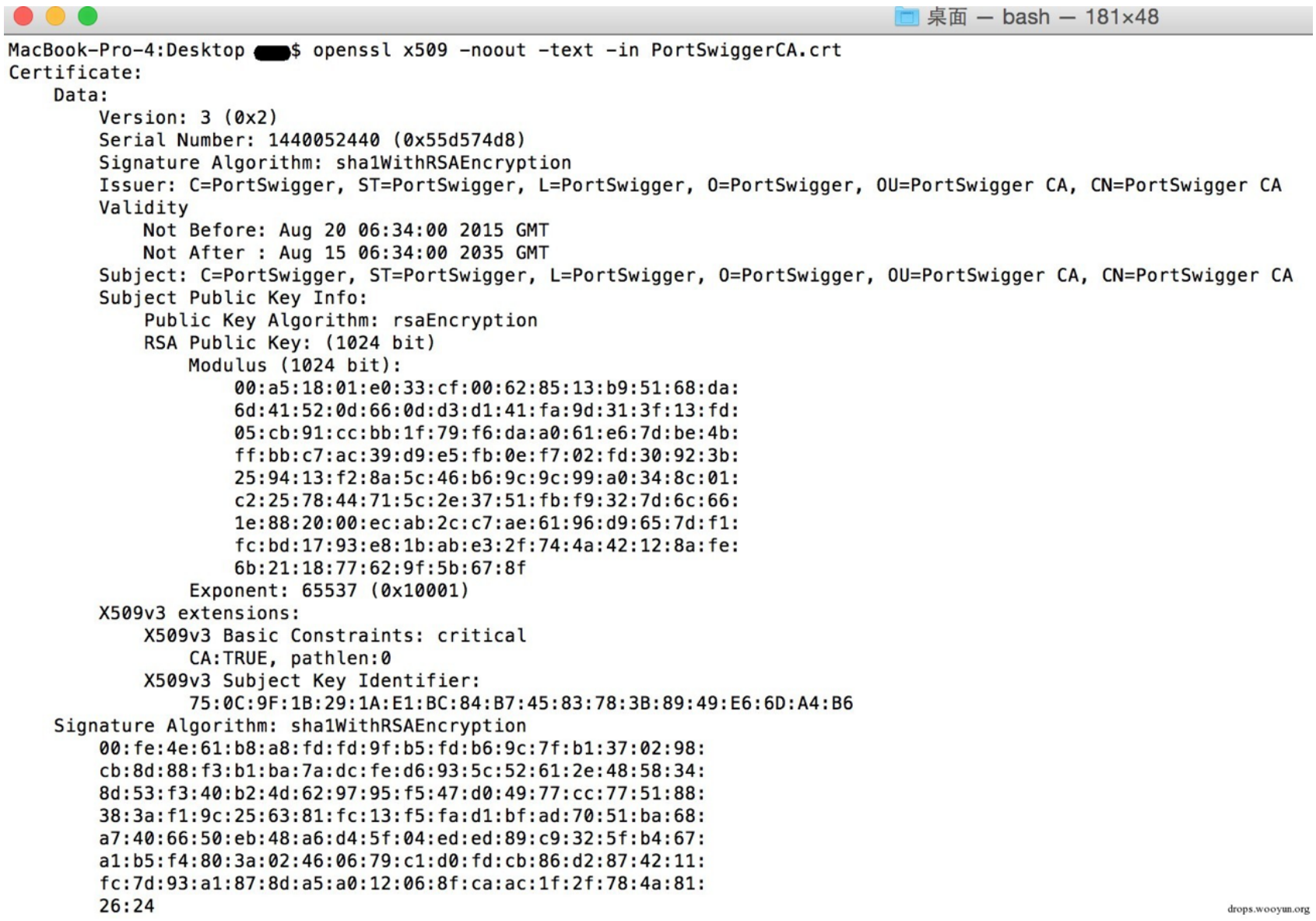
如果客户端对证书链做了校验，那么攻击难度就会上升一个层次，此时需要人为的信任伪造的证书或者安装伪造的CA公钥证书从而间接信任伪造的证书，可以使用burp的如下模块进行中间人攻击。





4.手机客户端Https数据包抓取

上述第一、二种情况不多加赘述，第三种情况就是我们经常使用的抓手机应用https数据包的方法，即导入代理工具的公钥证书到手机里，再进行https数据包的抓取。导入手机的公钥证书在android平台上称之为受信任的凭据，在ios平台上称之为描述文件，可以通过openssl的命令直接查看我们导入到手机客户端里的这个PortSwiggerCA.crt。

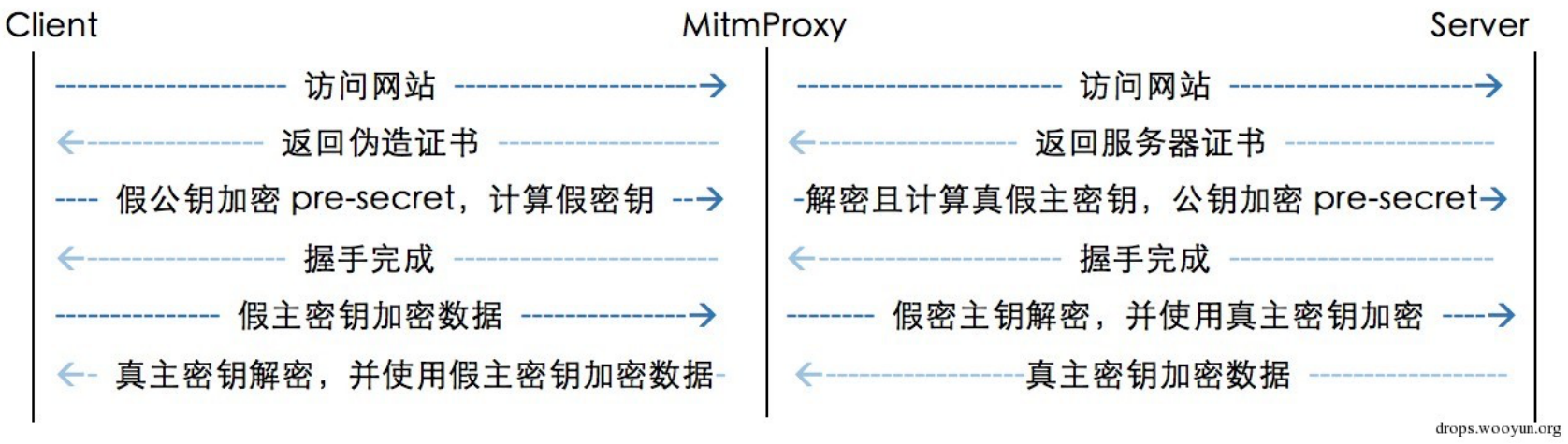


可以看见是Issuer和Subject一样的自签名CA公钥证书，另外我们也可以通过证书类型就可以知道此为公钥证书，crt、der格式的证书不支持存储私钥或证书路径(有兴趣的同学可查找证书相关信息)。导入CA公钥证书之后，参考上文的证书校验过程不难发现通过此方式能通过证书链校验，从而形成中间人攻击，客户端使用代理工具的公钥证书加密随机数，代理工具使用私钥解密并计算得到对称加密密钥，再对数据包进行解密即可抓取明文数据包。

5.中间人攻击原理

一直在说中间人攻击，那么中间人攻击到底是怎么进行的呢，下面我们通过一个流行的MITM开源库mitmproxy来分析中间人攻击的原理。中间人攻击的关键在于https握手过程的ClientKeyExchange，由于pre key交换的时候是使用服务器证书里的公钥进行加密，如果用的伪造证书的公钥，那么中间人就

可以解开该密文得到pre_master_secret计算出用于对称加密算法的master_key，从而获取到客户端发送的数据;然后中间人代理工具再使用其和服务端的master_key加密传输给服务端;同样的服务器返回给客户端的数据也是经过中间人解密再加密，于是完整的https中间人攻击过程就形成了，一图胜千言，来吧。



通过读Mitmproxy的源码发现mitmproxy生成伪造证书的函数如下：

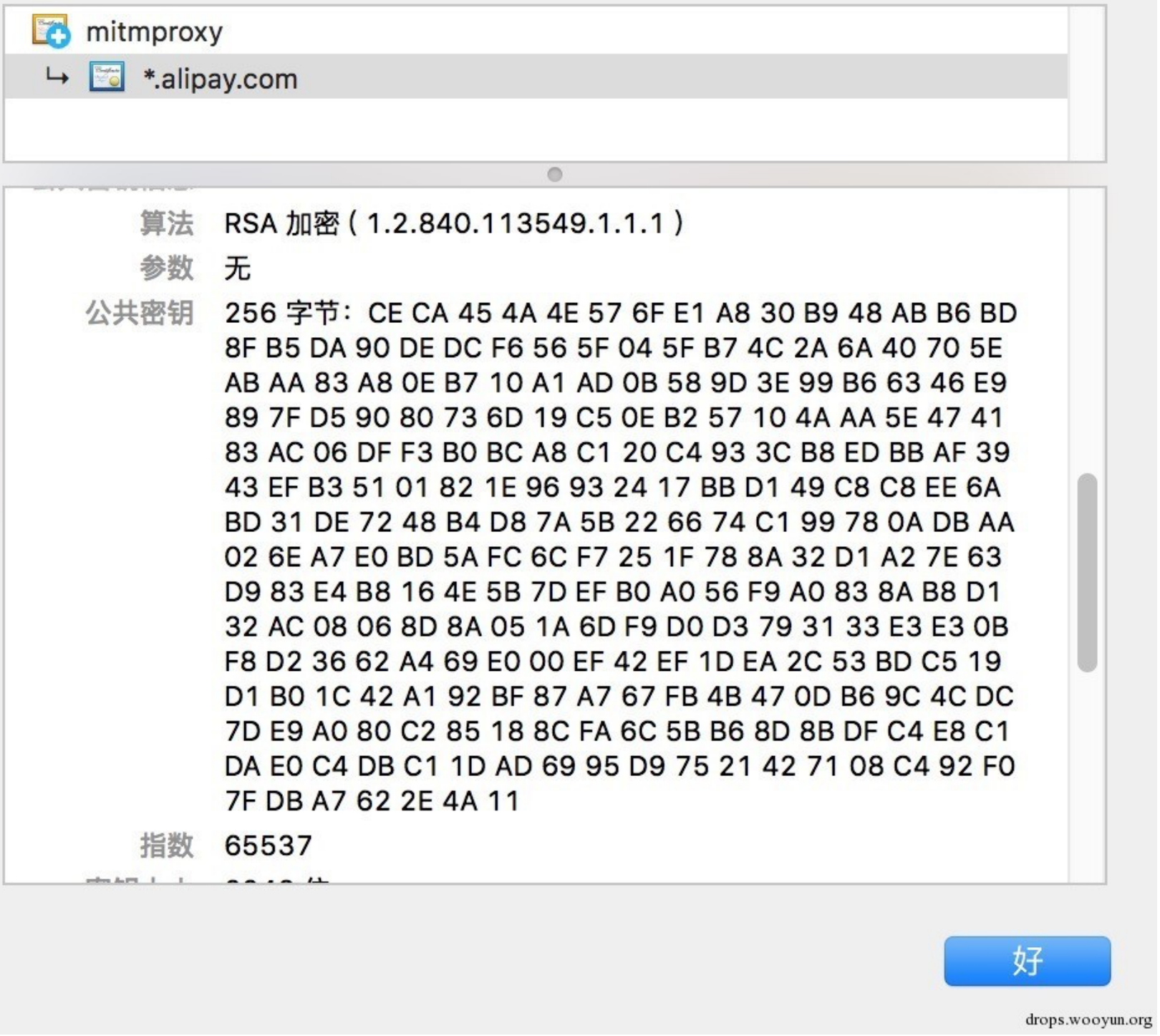
```
def dummy_cert(privkey, cacert, commonname, sans):
    """
        Generates a dummy certificate.

        privkey: CA private key
        cacert: CA certificate
        commonname: Common name for the generated certificate.
        sans: A list of Subject Alternate Names.

        Returns cert if operation succeeded, None if not.
    """
    ss = []
    for i in sans:
        try:
            ipaddress.ip_address(i.decode("ascii"))
        except ValueError:
            ss.append(b"DNS: %s" % i)
        else:
            ss.append(b"IP: %s" % i)
    ss = b", ".join(ss)

    cert = OpenSSL.crypto.X509()
    cert.gmtime_adj_notBefore(-3600 * 48)
    cert.gmtime_adj_notAfter(DEFAULT_EXP)
    cert.set_issuer(cacert.get_subject()) 将伪造证书的颁发者设置为mitm证书的使用者，可通过证书链校验
    if commonname is not None:
        cert.get_subject().CN = commonname 设置伪造证书的使用者，直接使用服务器证书的原始数据
    cert.set_serial_number(int(time.time() * 10000))
    if ss:
        cert.set_version(2)
        cert.add_extensions(
            [OpenSSL.crypto.X509Extension(b"subjectAltName", False, ss)])
    cert.set_pubkey(cacert.get_pubkey()) 设置伪造证书的公钥为mitm的公钥，后续可使用对应私钥解密数据
    cert.sign(privkey, "sha256") 通过sha256算法计算证书摘要，并使用mitm私钥对摘要进行加密形成签名
    return SSLCert(cert)
```


通过上述函数一张完美伪造的证书就出现了，使用浏览器通过mitmproxy做代理看下实际伪造出来的证书。



可以看到实际的证书是由mimtproxy颁发的，其中的公钥就是mimtproxy自己的公钥，后续的加密数据就可以使用mimtproxy的私钥进行解密了。如果导入了mitmproxy的公钥证书到客户端，那么该伪造的证书就可以完美的通过客户端的证书校验了。这就是平时为什么导入代理的CA证书到手机客户端能抓取https的原因。

四、证书校验

通过上文第一和第二部分的说明，相信大家已经对https有个大概的了解了，那么问题来了，怎样才能防止这些“中间人攻击”呢?

app证书校验已经是一个老生常谈的问题了，但是市场上还是有很多的app未做好证书校验，有些只做了部分校验，例如检查证书域名是否匹配证书是否过期，更多数的是根本就不做校验，于是就造成了中间人攻击。做证书校

验需要做完全，只做一部分都会导致中间人攻击，对于安全要求并不是特别高的app可使用如下校验方式：

- 查看证书是否过期
- 服务器证书上的域名是否和服务器的实际域名相匹配
- 校验证书链

可参考http://drops.wooyun.org/tips/3296，此类校验方式虽然在导入CA公钥证书到客户端之后会造成中间人攻击，但是攻击门槛已相对较高，所以对于安全要求不是特别高的app可采用此方法进行防御。对于安全有较高要求一些app(例如金融)上述方法或许还未达到要求，那么此时可以使用如下更安全的校验方式，将服务端证书打包放到app里，再建立https链接时使用本地证书和网络下发证书进行一致性校验。

此类校验即便导入CA公钥证书也无法进行中间人攻击，但是相应的维护成本会相对升高，例如服务器证书过期，证书更换时如果app不升级就无法使用，那么可以改一下，生成一对RSA的公私钥，公钥可硬编码在app，私钥放服务器。 https握手前可通过服务器下发证书信息，例如公钥、办法机构、签名等，该下发的信息使用服务器里的私钥进行签名; 通过app里预置的公钥验签得到证书信息并存在内容中供后续使用; 发起https连接获取服务器的证书，通过对比两个证书信息是否一致进行证书校验。

这样即可避免强升的问题，但是问题又来了，这样效率是不是低太多了?答案是肯定的，所以对于安全要求一般的应用使用第一种方法即可，对于一些安全要求较高的例如金融企业可选择第二种方法。

说了挺多，但是该来的问题还是会来啊！现在的app一般采用混合开发，会使用很多webveiw直接加载html5页面，上面的方法只解决了java层证书校验的问题，并没有涉及到webview里面的证书校验，对于这种情况怎么办呢?既然问题来了那么就一起说说解决方案，对于webview加载html5进行证书校验的方法如下：

webview创建实例加载网页时通过onPageStart方法返回url地址; 将返回的地址转发到java层使用上述的证书校验代码进行进行校验; 如果证书校验出错则使用stoploading()方法停止网页加载，证书校验通过则正常加载。