

安装：

pip3 install mitmproxy

证书配置：

首先，运行下面这个命令产生CA证书，并启动mitmdump

mitmdump

接下来我们就可以在用户目录下的 .mitmproxy 目录下找到 CA 证书，如下：

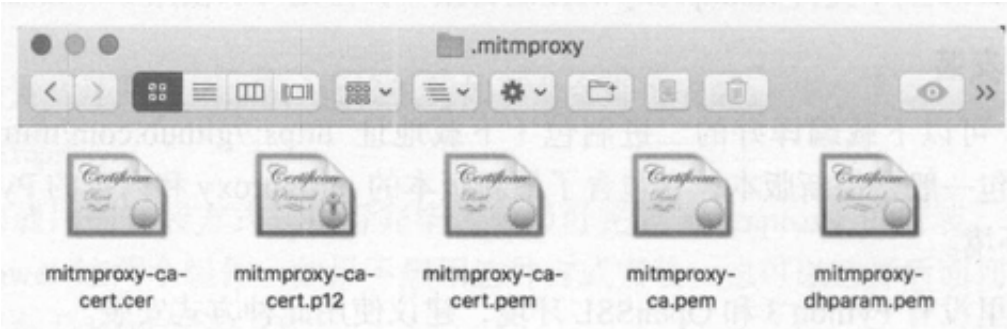


表 1-1 5 个证书及其说明

名 称	描 述
mitmproxy-ca.pem	PEM 格式的证书私钥
mitmproxy-ca-cert.pem	PEM 格式证书，适用于大多数非 Windows 平台
mitmproxy-ca-cert.p12	PKCS12 格式的证书，适用于 Windows 平台
mitmproxy-ca-cert.cer	与 mitmproxy-ca-cert.pem 相同，只是改变了后缀，适用于部分 Android 平台
mitmproxy-dhparam.pem	PEM 格式的秘钥文件，用于增强 SSL 安全性

● Mac

Mac 下双击 mitmproxy-ca-cert.pem 即可弹出钥匙串管理页面，然后找到 mitmproxy 证书，打开其设置选项，选择“始终信任”即可，如图 1-64 所示。

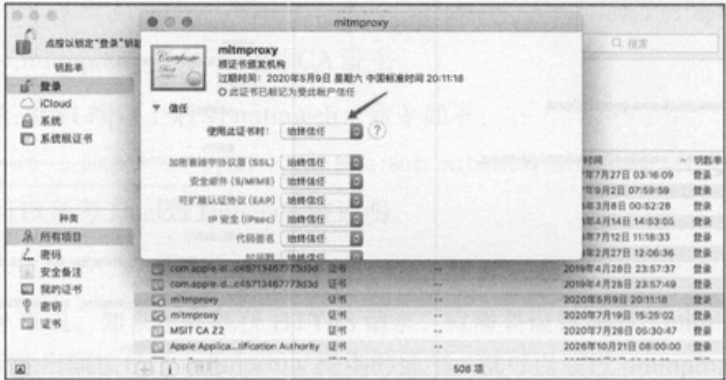


图 1-64 证书配置

● iOS

将 mitmproxy-ca-cert.pem 文件发送到 iPhone 上，推荐使用邮件方式发送，然后在 iPhone 上可以直接点击附件并识别安装，如图 1-65 所示。

点击“安装”按钮之后，会跳到安装描述文件的页面，点击“安装”按钮，此时会有警告提示，如图 1-66 所示。

继续点击右上角的“安装”按钮，安装成功之后会有已安装的提示，如图 1-67 所示。

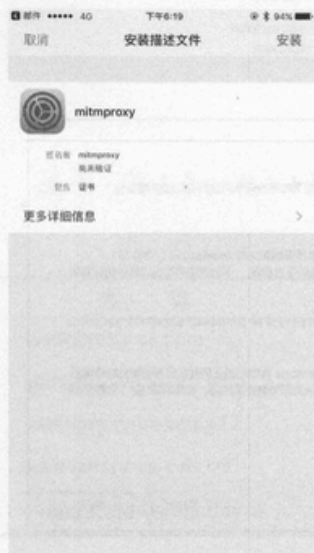


图 1-65 证书安装页面

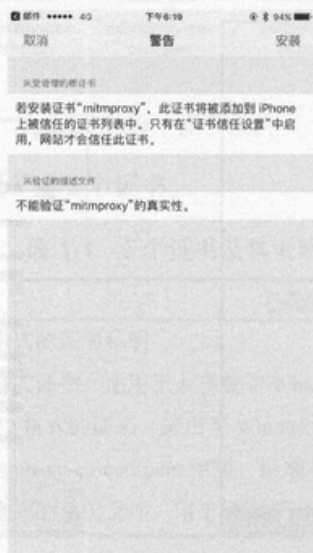


图 1-66 安装警告页面

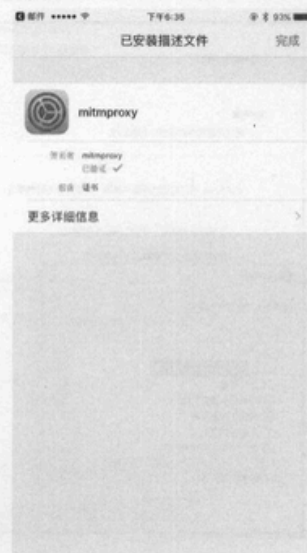


图 1-67 安装成功页面

如果你的 iOS 版本是 10.3 以下的话，此处信任 CA 证书的流程就已经完成了。

如果你的 iOS 版本是 10.3 及以上版本，还需要在“设置”→“通用”→“关于本机”→“证书信任设置”将 mitmproxy 的完全信任开关打开，如图 1-68 所示。此时，在 iOS 上配置信任 CA 证书的流程就结束了。

此时可以先将运行着 mitmdump 的 terminal 退出

首先，我们需要运行 mitmproxy，命令如下所示：

启动 mitmproxy 的命令如下：

```
mitmproxy
```

之后会在 8080 端口上运行一个代理服务，如图 11-12 所示。

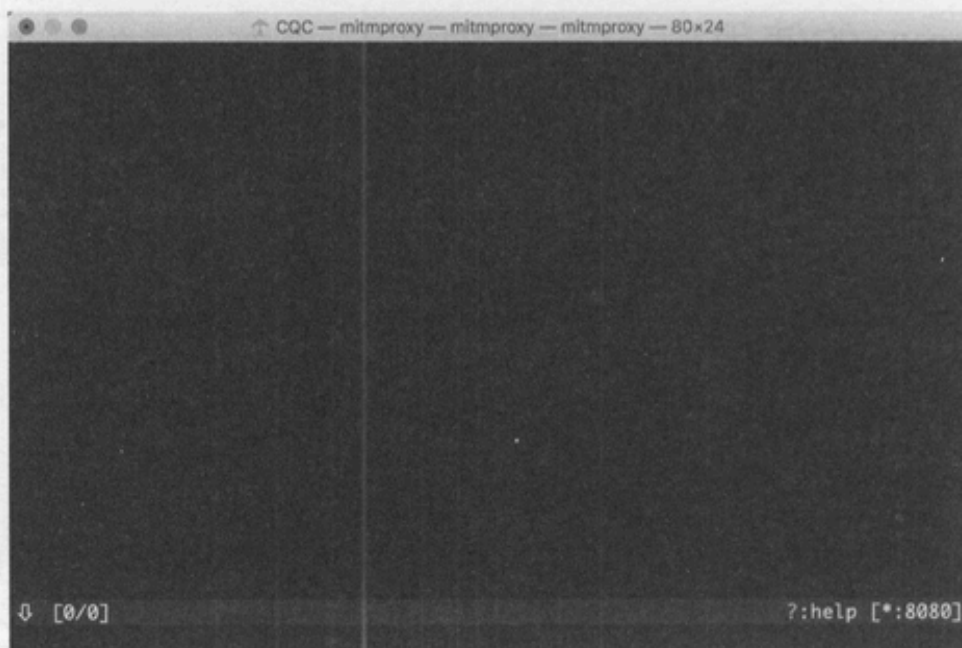


图 11-12 mitmproxy 运行结果

右下角会出现当前正在监听的端口。

现在我们需要把 **手机**和 **PC** 连接在同一局域网下，设置代理为当前代理，首先查看 PC 当前局域网的 IP

运行：

```
ifconfig
```

```
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
    ether f0:79:60:01:8f:fa
    inet6 fe80::c7c:e222:ce45:ec1b%en0 prefixlen 64 secured scopeid 0x4
    inet 10.201.8.196 netmask 0xfffffe00 broadcast 10.201.9.255
    nd6 options=201<PERFORMNUD,DAD>
    media: autoselect
    status: active
```

一般类似 10.*.*或 172.16.*.*或 192.168.1.*这样的 IP 就是当前 PC 的局域网 IP，例如此图中 PC

的 IP 就是 10.201.8.196，然后把手机代理设置成



图 11-15 代理设置

这个图不是自己的，要把手动里面的服务器地址设置成 **10.201.8.196**

这样我们就配置好了 mitmproxy 的代理

Fiddler、Charles 也有这个功能，而且它们的图形界面操作更加方便。那么 mitmproxy 的优势何在？

mitmproxy 的强大之处体现在它的另一个工具 mitmdump，有了它我们可以直接对接 Python 对请求进行处理。下面我们来看看 mitmdump 的用法。

6. mitmdump 的使用

mitmdump 是 mitmproxy 的命令行接口，同时还可以对接 Python 对请求进行处理，这是相比 Fiddler、Charles 等工具更加方便的地方。有了它我们可以不用手动截获和分析 HTTP 请求和响应，只需写好请求和响应的处理逻辑即可。它还可以实现数据的解析、存储等工作，这些过程都可以通过 Python 实现。

● 实例引入

我们可以使用命令启动 mitmproxy，并把截获的数据保存到文件中，命令如下所示：

```
mitmdump -w outfile
```

其中 outfile 的名称任意，截获的数据都会被保存到此文件中。

还可以指定一个脚本来处理截获的数据，使用 -s 参数即可：

```
mitmdump -s script.py
```

这里指定了当前处理脚本为 script.py，它需要放置在当前命令执行的目录下。

我们可以在脚本里写入如下的代码：

```
def request(flow):
    flow.request.headers['User-Agent'] = 'MitmProxy'
    print(flow.request.headers)
```

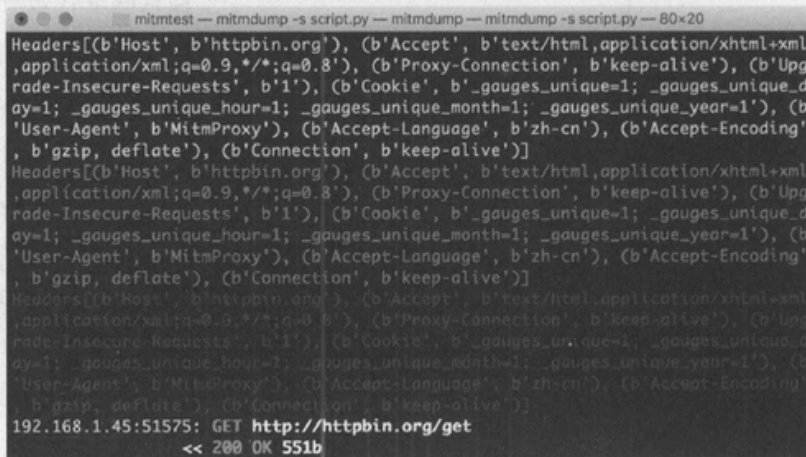
● 日志输出

mitmdump 提供了专门的日志输出功能，可以设定不同级别以不同颜色输出结果。我们把脚本修改成如下内容：

```
from mitmproxy import ctx

def request(flow):
    flow.request.headers['User-Agent'] = 'MitmProxy'
    ctx.log.info(str(flow.request.headers))
    ctx.log.warn(str(flow.request.headers))
    ctx.log.error(str(flow.request.headers))
```

这里调用了 ctx 模块，它有一个 log 功能，调用不同的输出方法就可以输出不同颜色的结果，以方便我们做调试。例如，info()方法输出的内容是白色的，warn()方法输出的内容是黄色的，error()方法输出的内容是红色的。运行结果如图 11-26 所示。



```
mitmtest -- mitmdump -s script.py -- mitmdump -- mitmdump -s script.py -- 80x20
Headers[(b'Host', b'httpbin.org'), (b'Accept', b'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8'), (b'Proxy-Connection', b'keep-alive'), (b'Upgrade-Insecure-Requests', b'1'), (b'Cookie', b'_gauges_unique=1; _gauges_unique_day=1; _gauges_unique_hour=1; _gauges_unique_month=1; _gauges_unique_year=1'), (b'User-Agent', b'MitmProxy'), (b'Accept-Language', b'zh-cn'), (b'Accept-Encoding', b'gzip, deflate'), (b'Connection', b'keep-alive')]
Headers[(b'Host', b'httpbin.org'), (b'Accept', b'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8'), (b'Proxy-Connection', b'keep-alive'), (b'Upgrade-Insecure-Requests', b'1'), (b'Cookie', b'_gauges_unique=1; _gauges_unique_day=1; _gauges_unique_hour=1; _gauges_unique_month=1; _gauges_unique_year=1'), (b'User-Agent', b'MitmProxy'), (b'Accept-Language', b'zh-cn'), (b'Accept-Encoding', b'gzip, deflate'), (b'Connection', b'keep-alive')]
Headers[(b'Host', b'httpbin.org'), (b'Accept', b'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8'), (b'Proxy-Connection', b'keep-alive'), (b'Upgrade-Insecure-Requests', b'1'), (b'Cookie', b'_gauges_unique=1; _gauges_unique_day=1; _gauges_unique_hour=1; _gauges_unique_month=1; _gauges_unique_year=1'), (b'User-Agent', b'MitmProxy'), (b'Accept-Language', b'zh-cn'), (b'Accept-Encoding', b'gzip, deflate'), (b'Connection', b'keep-alive')]
192.168.1.45:51575: GET http://httpbin.org/get
<< 200 OK 551b
```

图 11-26 运行结果

不同的颜色对应不同级别的输出，我们可以将不同的结果合理划分级别输出，以更直观方便地查看调试信息。

同时我们还可以对任意属性进行修改，就像最初修改 Headers 一样，直接赋值即可。例如，这里将请求的 URL 修改一下，脚本修改如下所示：

```
def request(flow):
    url = 'https://httpbin.org/get'
    flow.request.url = url
```

比较有意思的是，浏览器最上方还是呈现百度的 URL，但是页面已经变成了 httpbin.org 的页面了。另外，Cookies 明显还是百度的 Cookies。我们只是用简单的脚本就成功把请求修改为其他的站点。通过这种方式修改和伪造请求就变得轻而易举。

通过这个实例我们知道，有时候 URL 虽然是正确的，但是内容并非是正确的。我们需要进一步提高自己的安全防范意识。

Request 还有很多属性，在此不再一一列举。更多属性可以参考：<http://docs.mitmproxy.org/en/latest/scripting/api.html>。

爬去数据的时候直接取看崔大的代码或者网上搜～～