

网络安全工程实践：实验五报告

计 75 班 赵成钢 2017011362

计 75 班 顾煜贤 2017011421

2020 年 12 月

目录

1 代码使用方式	2
1.1 远程服务器	2
1.2 本地服务器	2
1.3 文件格式	2
1.4 挂载代理	2
2 协议设计	4
2.1 整体逻辑	4
2.2 具体协议格式	5
2.2.1 握手	5
2.2.2 通信	6
2.3 SOCKS5 代理实现和解析	6
2.4 实现方案	7
2.4.1 并发实现	7
2.4.2 加密实现	7
3 安全性分析	7
3.1 流量监测	7
3.2 数据窃取	7
3.3 阻断	8
4 功能测试截图	8
5 分工	10

1 代码使用方式

1.1 远程服务器

将 Wazi 目录放在代理服务器上。服务器端运行需要指定部署的端口，RSA 私钥地址以及密码文件存储的地址。在 Wazi 目录下，命令行运行以下命令：

```
python3 server.py
  -p SERVER_PORT # 代理服务部署的端口，默认为 8388
  -k PRIVATE_KEY # 私钥路径，默认为 private_key
  -pw PASSWORD_PATH # 用户名密码文件路径，默认为 password
```

成功运行后，程序会监听在指定的端口，等待连接。如果这时有客户端发起连接，并且通过认证，则服务器端会有如下输出：

```
[root@VM-8-15-centos Wazi] python3 server.py
Remote server starts running
Listening at 0.0.0.0:8388
> Correct Password and exchange AES key with 59.66.131.34
> Authentication OK
```

此时服务器端和客户端开始通信。

1.2 本地服务器

在 Wazi 目录中运行 local.py，指定部署的地址和端口位置、远程服务器地址、端口和公钥地址以及用户名密码，即可向服务器发起连接，如果中途客户端退出，可以再次启动再次和服务器认证；如果中途服务器端退出，则客户端需要手动重新启动并和服务器握手。具体命令如下：

```
python3 local.py
  -sa SERVER_ADDRESS # 远程服务器地址， 默认为 106.52.13.105
  -sp SERVER_PORT # 远程服务器地址端口， 默认为 8388
  -la LOCAL_ADDRESS # 本地服务监听地址， 默认为 localhost
  -lp LOCAL_PORT # 本地服务监听端口， 默认为 1080
  -u USER # 用户名， 默认为 admin
  -p PASSWORD # 密码， 默认为 admin
  -k PUBLIC_KEY_ADDRESS # 服务器公钥路径或 HTTPS 地址， 默认为 public_key
```

1.3 文件格式

私钥文件有两行，分别对应 RSA 中 n 和 d；公钥文件有两行，分别对应 RSA 中 n 和 e；password 文件有两行，分别为用户名和密码。

在代码目录中，我们也提供了示例的三个文件。

1.4 挂载代理

在 macOS 系统中，可以通过网络设置中的 SOCK5 代理（输入上文的本地服务器地址）来挂载全局 SOCK5 代理，挂载完成后即可全局使用代理，如下图所示。

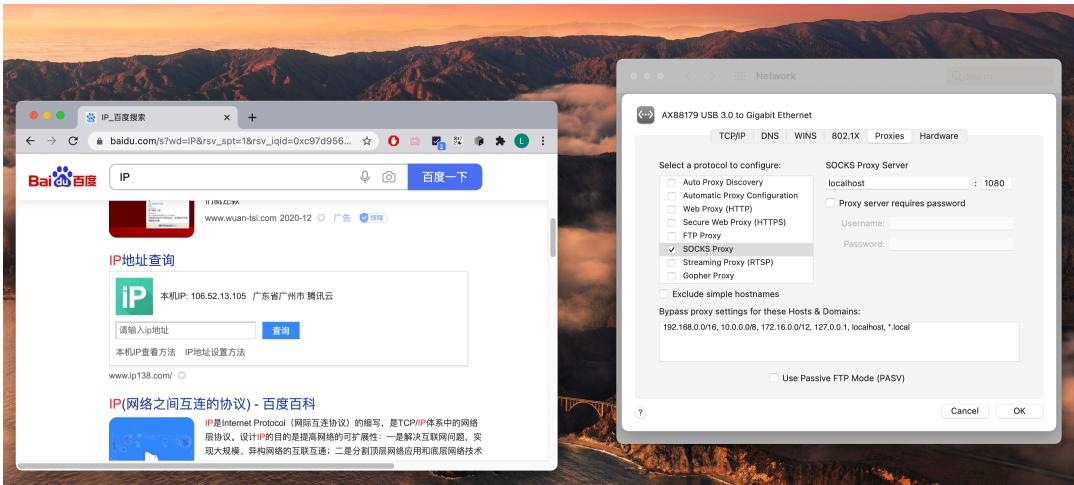


图 1: 全局用腾讯云代理并查询 IP 地址

在其他拥有命令行的系统中，也可以通过环境变量来指定 HTTP 和 HTTPS 的代理，具体命令如下：

```
export http_proxy="socks5://localhost:1080" # 引号中为地址
export https_proxy="socks5://localhost:1080" # 引号中为地址
```

随后可以通过 curl 等命令访问网站，如下图所示。

```
lyricz@Lyrics-MacBook-Pro:~ Last login: Sat Dec 12 16:21:24 on ttys002 → ~ export http_proxy="socks5://localhost:1080" → ~ curl www.baidu.com <!DOCTYPE html> <html> <head><meta http-equiv=content-type content=text/html; charset=utf-8><meta http-equiv=X-UA-Compatible content=IE=Edge><meta content=always name=referrer><link rel=stylesheet type=text/css href=http://s1.bdstatic.com/r/www/cache/bdorz/baidu.min.css><title>百度一下，你就知道</title></head> <body link=#0000cc> <div id=wrapper> <div id=head> <div class=head_wrapper> <div class=s_form> <div class=s_form_wrapper> <div id=lg> <img hidefocus=true src=/www.baidu.com/img/bd_logo1.png width=270 height=129> </div> <form id=form name=f action=/www.baidu.com/s class=fm> <input type=hidden name=bdorz_come value=1> <input type=hidden name=ie value=utf-8> <input type=hidden name=f value=8> <input type=hidden name=rsv_bp value=1> <input type=hidden name=rsv_idx value=1> <input type=hidden name=tn value=baidu><span class="bg_s_ipr_wr"><input id=kw name=wd class=s_ipr value=maxlength=255 autocomplete=off autofocus></span><span class="bg_s_btn _wr"><input type=submit id=su value=百度一下 class="bg_s_btn"></span> </form> </div> </div> <div id=u1> <a href=http://news.baidu.com name=tj_trnews class=mnav>新闻</a> <a href=http://www.hao123.com name=tj_trhao123 class=mnav>hao123</a> <a href=http://map.baidu.com name=tj_trmap class=mnav>地图</a> <a href=http://v.baidu.com name=tj_trvideo class=mnav>视频</a> <a href=http://tieba.baidu.com name=tj_trtieba class=mnav>贴吧</a> <noscript> <a href=http://www.baidu.com/bdorz/login.gif?login&tpl=mn&u=http%3A%2F%2Fwww.baidu.com%2f%3fbdorz_come%3d1 name=tj_login class=lb>登录</a> </noscript> <script>document.write('<a href='http://www.baidu.com/bdorz/login.gif?login&tpl=mn&u='+ encodeURIComponent(window.location.href)+ '>'+ decodeURIComponent(window.location.href)+ '</a>');</script>
```

图 2: 使用 curl 命令访问百度

在其他操作系统或浏览器中，如果想用全局代理，还可以为 Chrome、FireFox 等浏览器安装 SOCK5 代理插件，随后进行输入对应的本地服务器的地址既可代理上网。

2 协议设计

2.1 整体逻辑

本实验的很多设计从攻击者出发，具体代理的协议其实并不重要，最重要的是加密后的信道，所以对于代理协议，我们选择了最简单的 SOCK5，后面我们本地服务器和远程服务器的通信是我们自己设计的协议，如下图所示。

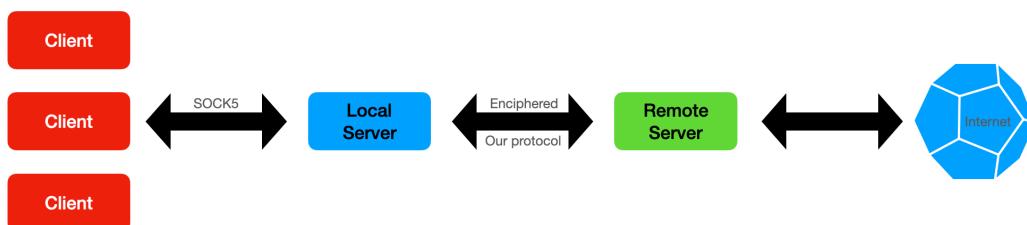


图 3: 整体通信方式

后面，我们的设计从不容易“被发现”和“高效率”出发。首先，协议一定不是明文协议，这意味着必须协商通信的密钥，这个协商的过程，如果不用非对称加密，就会发生中间人伪造的攻击（除非物理上协商对称加密的密钥），所以在协商的过程中，我们必须用非对称加密。而在协商之后，考虑到效率，我们选择了对称加密算法 AES（非对称 RSA 比较慢），而且为了更高的安全性和加大攻击的难度，我们需要为每一个会话都随机协商一个 AES 密钥。

总结一下上面的结论就是协商每个会话 AES 密钥用 RSA，后面通信为了效率用 AES。

那么第一个要解决的问题是怎么拿到服务器公钥，如果没有 CA，这个地方还是会被中间人伪造公钥，所以这里我们认为客户之前在物理上有远程服务器的公钥，比如用 CA 认证的邮箱服务商接受远程服务器的公钥，或者，其实直接把远程服务器的公钥放在一个 HTTPS 的网站上即可，在本实现中 local.py 的 PUBLIC_KEY_ADDRESS 参数就可以是一个 HTTPS 的地址，为了方便，我们把示例的公钥放到了 https://gitee.com/LyricZhao/Wazi/raw/main/public_key，该参数可以直接写这个地址也可以是一个文件路径。

那么既然有了公钥，这意味着不可能发生中间人挟持导致数据被解密，因为只有远程服务器的私钥可以解开加密的内容。

下一个问题是，如何去认证，这里最大的潜在风险是如果验证和后续的数据传输在协议中有一个明文的开头（如果不是明文，那只能用 RSA 和 AES 都计算一遍看看解密后的标记是不是符号认证或者传输，但是 RSA 效率很低，我们不希望这样做，这样容易减少了洪水攻击的难度）来让远程服务器区分是认证还是真的传输，这一定同样容易地被攻击者发现，后续被阻断。所以，我们的协议需要用一个非常隐晦的标记来告诉服务器是认证还是传输，才不会被发现。所以这里我们

的设计是用双因子认证的方式，如果是一个认证请求，需要把当前的时间窗口（以 10 秒为一个窗口）作为 seed 计算用户名和密码的 hash 值放到包头，同时 AES 的传输包不需要任何的包头（碰撞几乎不可能）。服务器每 10 秒会计算一遍所有用户（我们实现只写了一个用户的情况）的当前时间窗口的 hash 值，如果发来的包的包头命中了某一个用户，才会进一步验证真的包后面用 RSA 加密的用户名密码，如果全部命中，则把 RSA 加密的 AES 密钥设置给当前用户。后面的传输就全部用 AES，而且几乎不可能碰撞到前面的 hash，那不碰撞到就直接用高效的 AES 解密，这也绕开了低效率的 RSA 算法，而且可以通过非常晦涩的包头来区分到底是想要认证还是真的数据传输。

总结起来，整个流程如下图所示。

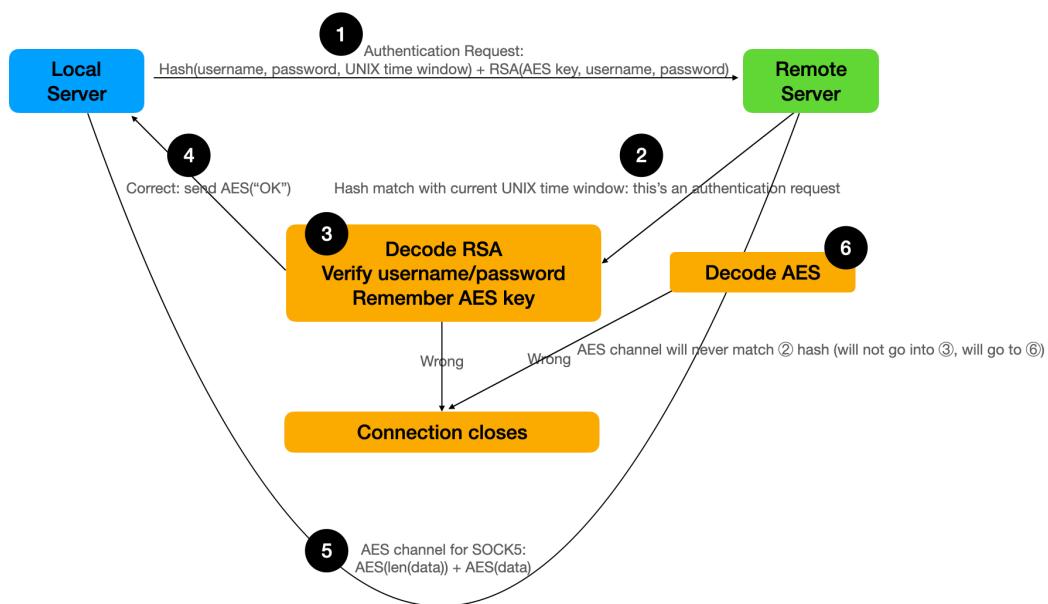


图 4: 整体流程

2.2 具体协议格式

2.2.1 握手

正如整体逻辑的章节所说，为了实现握手，数据包包括以时间为 seed 的用户名密码的 hash 值和经过 RSA 加密后的 AES key 和用户名密码，如下图所示。



图 5: 本地服务器向远程服务器发送的握手包

如果远程服务器发现包头前 4 字节命中当前窗口的 hash 缓存，就去解密 RSA，验证具体的

用户名和密码，并用密钥初始化 AES。

如果发现全部正确，则返回一个用 AES 加密后的 OK（如下所示），否则直接断开连接。

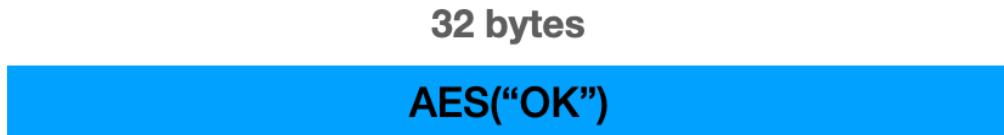


图 6: 远程服务器向本地服务器回复的成功

如果本地服务器想重新协商，只需要重新发送这个包并替换为新的 AES key，在协议中，我们规定每次成功的验证都会覆盖 AES key。

2.2.2 通信

有了协商之后的 AES 信道，把数据用 AES 加密之后发过去即可，不过考虑到 TCP 会发送分包和粘包等情况，我们需要规定一个长度。如果这个长度是明文，其实很容易变成一个识别协议的漏洞（比如我们规定 2 个字节是长度，那其实高字节很多时候会是 0）。所以，我们这里的做法是，这部分长度也用 AES 加密，我们把长度定义为 16 字节的整数，那么最后 AES 加密后长度就是固定的 32 位，也就是数据传输的包如下。



图 7: 本地服务器和远程服务器直接的通信

2.3 SOCKS5 代理实现和解析

浏览器通过本地服务器到远程服务器之间的通信使用 SOCKS5 协议。本地服务器和客户端（一般是浏览器）之间每建立一个连接，它都将同时和远程服务器端建立一个连接，用于转发数据。本地服务器和客户端之间的连接建立完成之后，客户端将持续发送 SOCKS5 数据包，这时本地服务器将用上述方式加密这些数据包，并将其转发到远程服务器端。本地服务器收到远程服务器端的回复后将解密数据包，发送给客户端。远程服务器端收到数据包后，会先解密这些数据包，然后解析 SOCKS5 协议，实现握手，得到用户希望访问的目的 IP 地址、端口号、待发送数据等信息，并和相应的目的服务器建立连接，再使用这些连接转发解析得到的用户数据。当服务器收到回复后，将数据包加密后发回给用户。

在 SOCKS5 协议中，通信两端需要进行两次握手，之后直接传输原始数据包。第一次握手中，客户端和服务器协商通信方式。客户端提供可选的通信方式列表，服务器端会判断是否为 SOCKS5 协议的握手信息，若判断成功，则回复数据选择某种特定的通信方式，这里我们采用“0x00”，即“不需要认证”。

第二次握手中，客户端发送的数据包中包含希望访问的地址、端口等信息。其中，地址可能为 IPv4 地址，域名或者 IPv6 地址。服务器收到这个数据包并解密完成后，将按照地址的不同种类分别和目的服务器建立连接，其中需要注意的是，如果地址为域名，则需要使用 `getaddrinfo(host, port)` 函数查找对应的 IP 地址。以上步骤完成之后，远程服务器将回复握手成功，此后本地服务器会发送浏览器的原始数据，由远程服务器转发至目的地址。

2.4 实现方案

2.4.1 并发实现

在实现方案上，我们整体采用了 Python 为语言，依托 Python 的 `async` 异步机制实现多连接的并发和维护。

具体代码参考 `local.py` 和 `server.py` 文件。

2.4.2 加密实现

本次作业我们实现了简易 RSA 算法，原理为输入对应的 n 、 e 、 d ，进行快速幂运算来加密和解密。

AES 算法由于比较复杂，我们调用了 `pycrypto` 库并实现了封装。

全部上述算法位于 `cipher.py` 文件中。

3 安全性分析

3.1 流量监测

本协议中为了绕开流量监测同时还能让服务器高效判断是握手还是通信，设计了上述的双因子 hash 的方式，私以为，在上述的全部包结构中没有一处明文，所以不存在数据上有特别明显的特征。

但是，其实可以注意到包的长度有一个相对较大的最小值，这里其实可以通过这点来进行一些可能的监测来攻击，不过其实也可以做随机的分片来绕过这点，这里我们没有实现。

另一个就是用户的行为特征，这点需要看用户为中心和谁在进行连接，然后结合一些用户的行为特征导致的流量特征来监测。

流量监测这点也是我认为比较难而且很重要的一个后续攻击的前提。

3.2 数据窃取

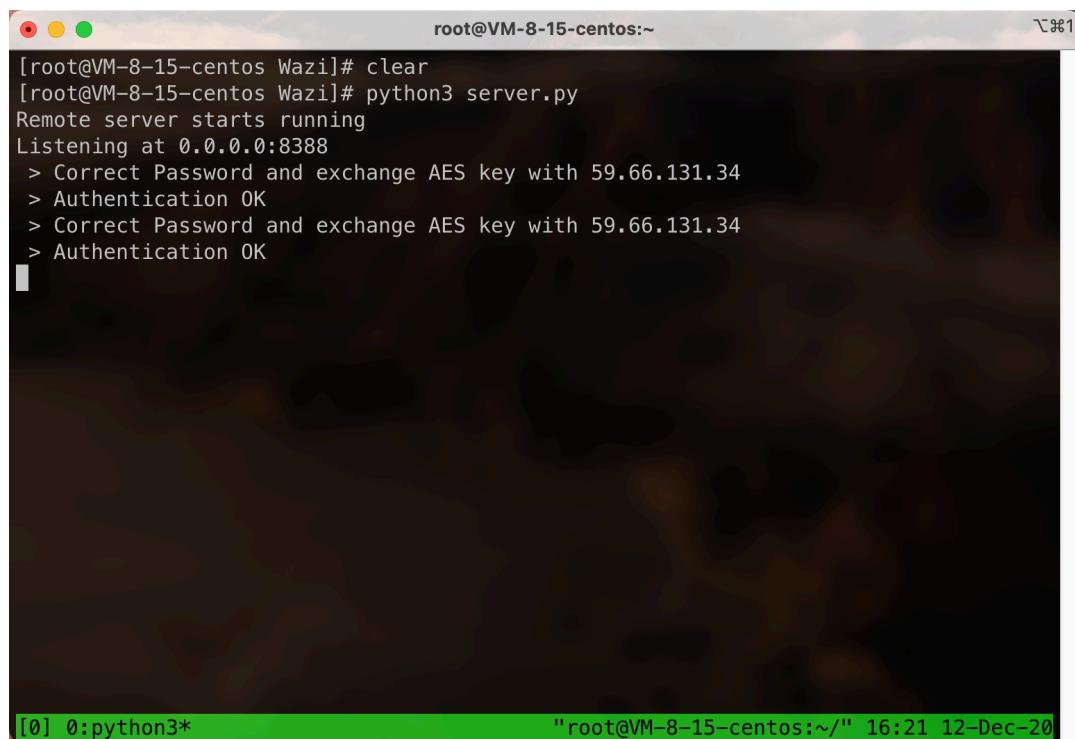
如果监测出来了目标地址，要进行数据的解密，唯一的手段就是拿到通信过程中的 AES key，如果这件事情只在信道上做（只能是中间人），那就只能伪造成远程服务器和本地服务器协商 AES key，但是本地会用公钥加密，所以除非拿到的公钥是错的，那么不可能窃取其中的数据。而为了不拿到错的公钥，这部分我们没有设计，但是一个很经典的问题，私以为在本实验中通过物理手段

或者 HTTPS 的其他可信第三方网站可以做到正确的公钥是一个大前提，所以不会有数据窃取的问题。

3.3 阻断

如果监测出来了目标地址，要进行阻断，其实只要不停发 TCP Reset 之类的包或者插入新的中间包来扰乱解密即可。这个地方我们认为不是我们协议的问题，任何一种这样的连接方式，一旦知道了目标，在一定程度都可以阻断（一个极端的例子是剪网线）。

4 功能测试截图



The screenshot shows a terminal window with a dark background and light-colored text. The title bar reads "root@VM-8-15-centos:~". The command entered is "python3 server.py". The output shows the server starting and listening on port 8388, followed by four log entries indicating successful password exchange and authentication with an IP address 59.66.131.34.

```
[root@VM-8-15-centos Wazi]# clear
[root@VM-8-15-centos Wazi]# python3 server.py
Remote server starts running
Listening at 0.0.0.0:8388
> Correct Password and exchange AES key with 59.66.131.34
> Authentication OK
> Correct Password and exchange AES key with 59.66.131.34
> Authentication OK
```

图 8: 远程服务器运行

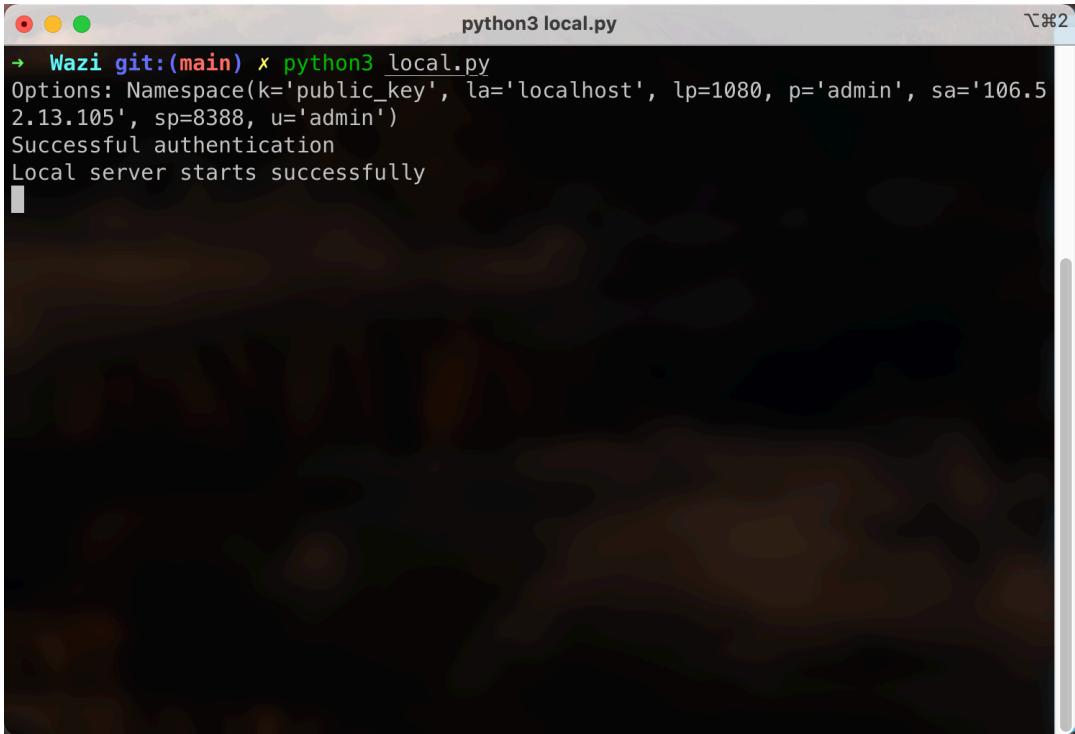


图 9: 本地服务器运行

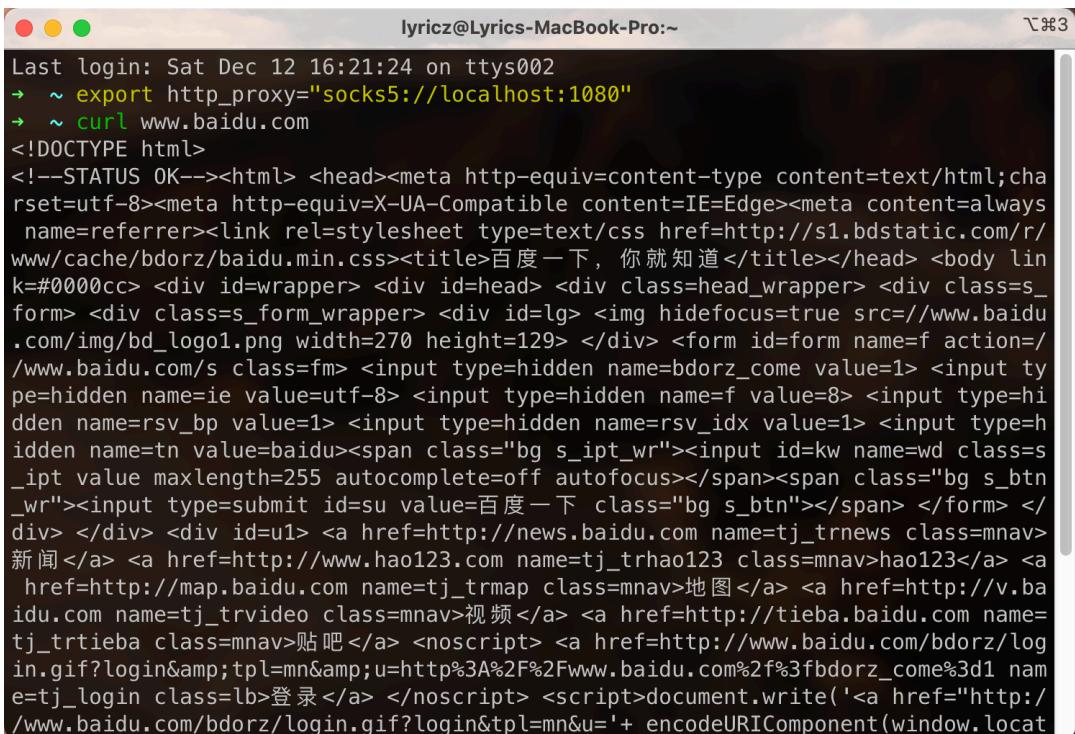


图 10: 使用 curl 命令访问百度

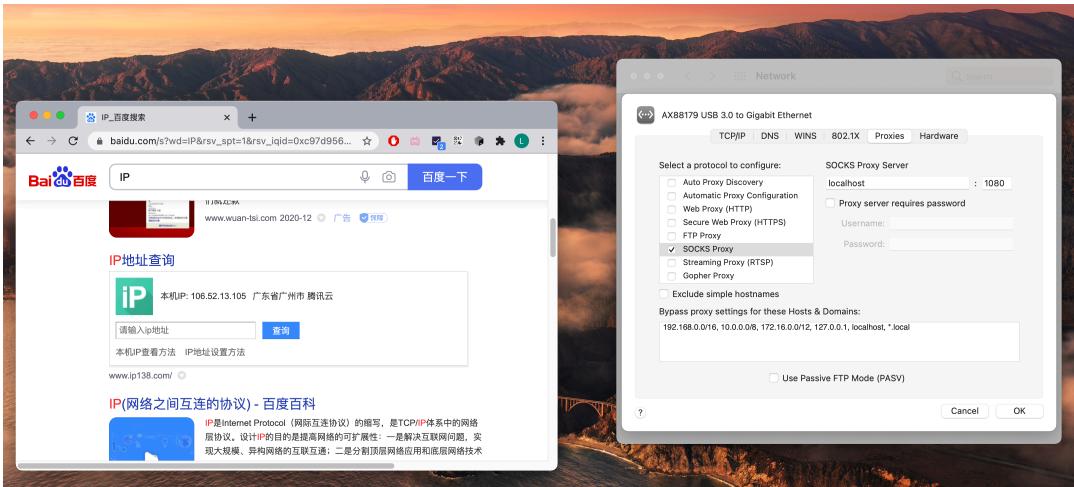


图 11: 全局用腾讯云代理并查询 IP 地址

5 分工

本组成员为赵成钢（2017011362）和顾煜贤（2017011421）。在本实验中，顾煜贤同学完成了远程服务器部分，赵成钢同学完成了本地服务器的部分。