Linux 口令破解及安全代理的设计和实现

1. 实验目标

学习 linux 下的密码加密方式，并学会利用工具进行给定加密密文的破解。

基于 SOCKS，利用密码算法设计一个安全的代理协议，具有认证、加密、完整性保护等功能。

2. 实验内容

- Linux 的口令破解

Xxx

- 安全代理的设计与实现

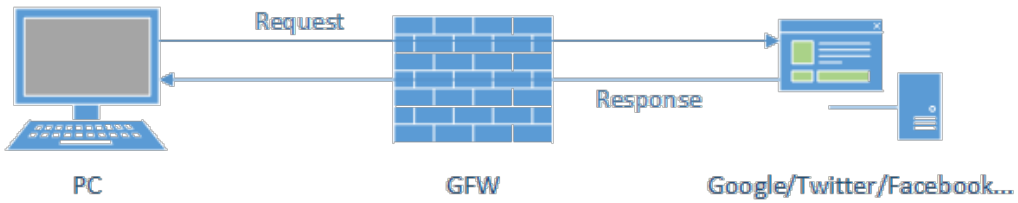
利用 socks5 协议以及 ssl 协议建立代理服务器，使浏览器的流量请求可以安全发送到目标 Web 服务器。

3. 实验原理

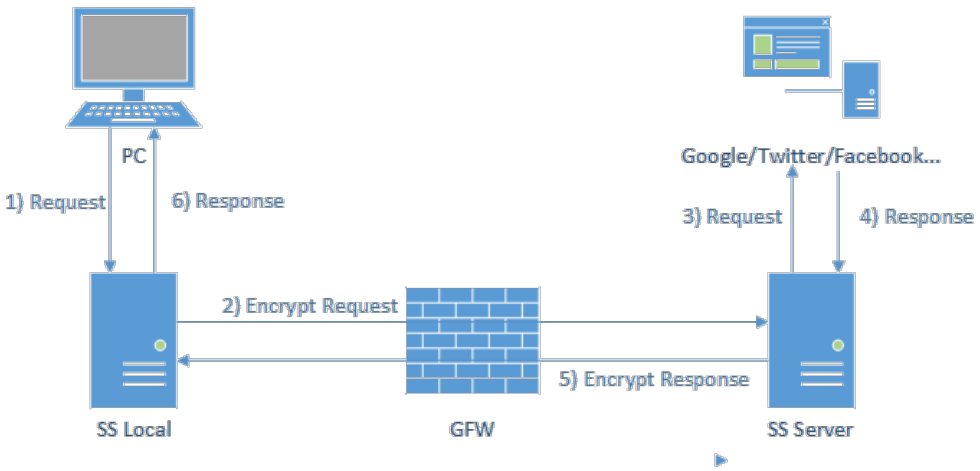
- Linux 口令格式

Xxxx

- shadowsocks 代理



目前我们使用浏览器访问 Web 服务器获取信息时，都需要经过 GFW，而 GFW 会将一些内容过滤，所以会导致我们有时候无法收到相应的响应数据。为了让我们的流量包不被 GFW 侦测拦截，clowwindy 提供了一种解决办法，如下图。



Shadowsocks 的工作流程如下：

1） 用户的浏览器发出请求，基于 socks5 协议与 ss-local 端通信。ss-local 一般部署在本地，所以不会被 GFW 用特征分析干扰。

2） ss-local 将用户浏览器的请求加密发送到境外的服务器 ss-server，由于流量包都经过加密，所以 GFW 无法对该流量包进行基于特征的过滤。

3） ss-server 接收到加密的流量后，进行解密，还原原来的请求，再发送到用户需要访问的目的服务器，获取响应。再加密传输给 ss-local。

4） ss-local 接收到 ss-server 发回的加密响应流量后，解密传输给用户，此时用户就可以获取到想要的访问结果。

- socks5 协议

socks5 协议主要分为 3 个阶段，认证阶段、连接阶段和传输阶段。

认证阶段

首先客户端需要和服务端握手认证，可以采用 *用户名/密码* 认证或者无认证方式。

格式如下：

```
+----+----------+----------+
|VER | NMETHODS | METHODS  |
+----+----------+----------+
| 1  |    1     |  1~255   |
+----+----------+----------+
```

VER - 当前协议版本号，5。 NMETHODS - METHODS 字段的字节数。

METHODS - 每一个字节为一种认证方法。常用 0x00：无认证，0x02: USERNAME/PASSWD。

服务器返回格式：

```
+----+--------+
|VER | METHOD |
```

```
+----+--------+
| 1  |   1    |
+----+--------+
```

VER - 当前协议版本号，5。

METHOD - 使用的认证方式，0xff：客户端任意一种认证方式，服务器都不支持。

若回复 0x05 0x02 表明用 *用户名/密码* 方式认证，需要按协议 RFC 1929 定义的方法进行认证。

连接阶段

认证完成后，客户端向服务器发送请求，格式如下：

```
 +----+-----+-------+------+----------+----------+
|VER | CMD |  RSV  | ATYP | DST.ADDR | DST.PORT |
+----+-----+-------+------+----------+----------+
| 1  |  1  |   1   |  1   | Variable |    2     |
+----+-----+-------+------+----------+----------+
```

VER - 当前协议版本，5。

CMD - command

0x01: CONNECT 建立 TCP 连接

0x02: BIND 上报反向连接地址

0x03: 关联 UDP 请求

RSV - 保留字段 0x00

ATYP - address type

0x01: IPv4

0x03: 域名

0x04: IPv6

DST.ADDR - destination address，取值随 ATYP 变化

DST.PORT - 目的服务器的端口

传输阶段

这一阶段，socks5 服务器只做单纯的转发功能

4. 实验分工

本实验中石英昊同学主要负责 linux 口令破解这一部分的内容，柴华君同学主要负责安全代理的内容。

5. 实验过程

- Linux 口令破解

- 安全代理的实现

此次实验安全代理主要参考了 shadowsock 的实现，并在其原有功能的基础上扩展了 socks5 协议的 *用户名/密码* 方法的认证，以及 ss local 与 ss server 之间通过 ssl 进行通信。

1. Socks5 协议的实现

首先配置主机使用 socks 代理，同时使用 *用户名/密码* 方法认证，用户名： test 密码： 123456

经过实际测试，Chrome 浏览器不支持这种认证方式，所以使用 mac 自带浏览器进行实验。

在本地运行 local.py 文件，代码实现如下图所示：

```
# auth stage  chaihj
header = self.rfile.read(2)
version, nmethod = struct.unpack("!2B", header)
assert version == 5
assert nmethod > 0
methods = []
logging.info("nmethod is %d" % nmethod)
for i in range(nmethod):
    method = ord(self.rfile.read(1))
    methods.append(method)
    logging.info("method : %d " % method)
if 2 not in methods:
    logging.error(" don't suport username/passwd auth ")
    return
self.wfile.write(struct.pack("!2B", 5, 2))  #use username/passwd auth
if not self.verify_credentials():
    return

# sock.recv(262)               # Sock5 Verification packet
sock.send("\x05\x00")        # Sock5 Response: '0x05' Version 5; '0x00' NO AUTHENTICATION REQUIRED
```

根据 RFC 1929 实现的用户认证代码实现

```python
def verify_credentials(self):
    version = ord(self.rfile.read(1))
    assert version == 1
    username_len = ord(self.rfile.read(1))
    username = self.rfile.read(username_len).decode('utf-8')
    passwd_len = ord(self.rfile.read(1))
    passwd = self.rfile.read(passwd_len).decode('utf-8')
    if username == self.username and passwd == self.password:
        response = struct.pack("!2B", version, 0)
        self.wfile.write(response)
        logging.info("username / passwd  auth  successfully")
        return True
    response = struct.pack("!2B", version, 0xFF)
    self.wfile.write(response)
    logging.info("username / passwd  auth  Failed")
    return False
```

当修改用户名时，认证失败：


```
2018-12-04 22:49:19 INFO     nmethod is 2
2018-12-04 22:49:19 INFO     method : 0
2018-12-04 22:49:19 INFO     method : 2
2018-12-04 22:49:19 INFO     username / passwd  auth  Failed
```

连接阶段，参考 sockets 协议和 shadowsocks 获取浏览器要访问的目标网站和目标端口。代码如下：

```
data = self.rfile.read(4)       # Forward request format: VER CMD RSV ATYP (4 bytes)
mode = ord(data[1])             # CMD == 0x01 (connect)
if mode != 1:
    logging.warn('mode != 1')
    return
addrtype = ord(data[3])         # indicate destination address type
logging.info("VER %d , CMD %d , RSV %d, ATYP %d" % (ord(data[0]), ord(data[1]), ord(data[2]), ord(data[3])))
addr_to_send = data[3]
if addrtype == 1:               # IPv4
    addr_ip = self.rfile.read(4)            # 4 bytes IPv4 address (big endian)
    addr = socket.inet_ntoa(addr_ip)
    addr_to_send += addr_ip
elif addrtype == 3:             # FQDN (Fully Qualified Domain Name)
    addr_len = self.rfile.read(1)           # Domain name's Length
    addr = self.rfile.read(ord(addr_len))   # Followed by domain name(e.g. www.google.com)
    addr_to_send += addr_len + addr
else:
    logging.warn('addr_type not support')
    return
addr_port = self.rfile.read(2)
addr_to_send += addr_port                   # addr_to_send = ATYP + [Length] + dst addr/domain name + port
port = struct.unpack('>H', addr_port)       # prase the big endian port number. Note: The result is a tuple ev
try:
    reply = "\x05\x00\x00\x01"              # VER REP RSV ATYP
    reply += socket.inet_aton('0.0.0.0') + struct.pack(">H", 2222)  # listening on 2222 on all addresses of th
    self.wfile.write(reply)                 # response packet
    # reply immediately
    if '-6' in sys.argv[1:]:                # IPv6 support
        remote = socket.socket(socket.AF_INET6, socket.SOCK_STREAM)
    else:
        remote = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    remote.setsockopt(socket.IPPROTO_TCP, socket.TCP_NODELAY, 1)       # turn off Nagling
    remote.connect((SERVER, REMOTE_PORT))
    ssl_remote = ssl.wrap_socket(remote, ca_certs="mycertfile.pem", cert_reqs=ssl.CERT_REQUIRED)
    ssl_remote.sendall(addr_to_send)
    logging.info('connecting %s:%d' % (addr, port[0]))
except socket.error, e:
    logging.info("socket error")
    logging.warn(e)
    return
self.handle_tcp(sock, ssl_remote)
```

访问百度：

Socks 服务器接收到目标服务器的域名和目标端口：

```
2018-12-05 16:01:33 INFO     VER 5 , CMD 1 , RSV 0, ATYP 3
2018-12-05 16:01:33 INFO     connecting www.baidu.com:443
2018-12-05 16:01:33 INFO     connecting sp0.baidu.com:443
2018-12-05 16:01:33 INFO     connecting sp2.baidu.com:443
2018-12-05 16:01:33 INFO     connecting sp1.baidu.com:443
2018-12-05 16:01:35 INFO     connecting hpd.baidu.com:443
```

2. 利用 ssl 进行流量加密转发

首先使用 openssl 工具产生自签名证书和私钥，命令如下：

openssl req -new -x509 -days 365 -nodes -out mycertfile.pem -keyout mykeyfile.pem

生成的证书和私钥文件如下图：

mykeyfile.pem

mycertfile.pem

使用证书和密钥来对用户流量进行加密，并同时对用户流量信息进行完整性检查。

代码如下：

```
ssl_remote = ssl.wrap_socket(remote, ca_certs="mycertfile.pem", cert_reqs=ssl.CERT_REQUIRED)
```

```
def handle_tcp(self, sock, ssl_remote):
    try:
        fdset = [sock, ssl_remote]
        while True:
            r, w, e = select.select(fdset, [], [])      # use select I/O multiplexing model
            if sock in r:                               # if local socket is ready for reading
                data = sock.recv(4096)
                if len(data) <= 0:                      # received all data
                    break
                ssl_remote.sendall(data)

            if ssl_remote in r:                             # remote socket(proxy) ready for reading
                data = ssl_remote.recv(4096)
                if len(data) <= 0:
                    break
                result = send_all(sock, data)
                if result < len(data):
                    raise Exception('failed to send all data')
    finally:
        sock.close()
        ssl_remote.close()
```

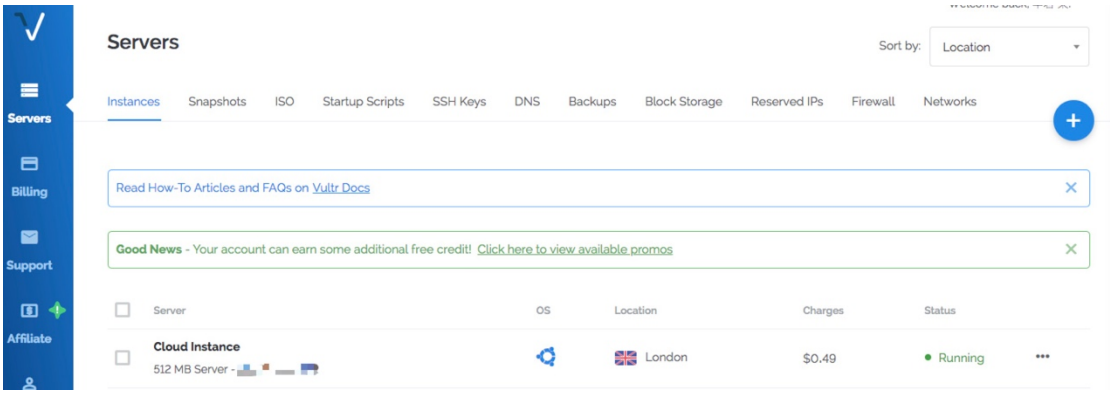
访问 google.com 检查是否能够绕过 GFW

成功访问 google 页面，表明代理成功绕过了 GFW 的拦截。

3. 境外服务器的设置

使用了 vultr 提供的位于 LONDON 的主机作为 ss server，具体的配置文件可查看附件中的 config.json 文件

境外服务器如下图：



6. 实验总结体会

这次实验让我们对于一些加密算法有了一定的了解，知道了如何去阅读 linux 的加密密文以及如何使用相应的工具进行密码爆破，感受到了一点点当黑客的乐趣。通过设计和实现安全代理，我们对于 socks5 的协议更加了解，对于常用的 shadowsocks 的原理也有了更深的认识。通过实验，真实地感受到了网络的魅力，希望能够学习更多的知识，去更好地探索网络世界。