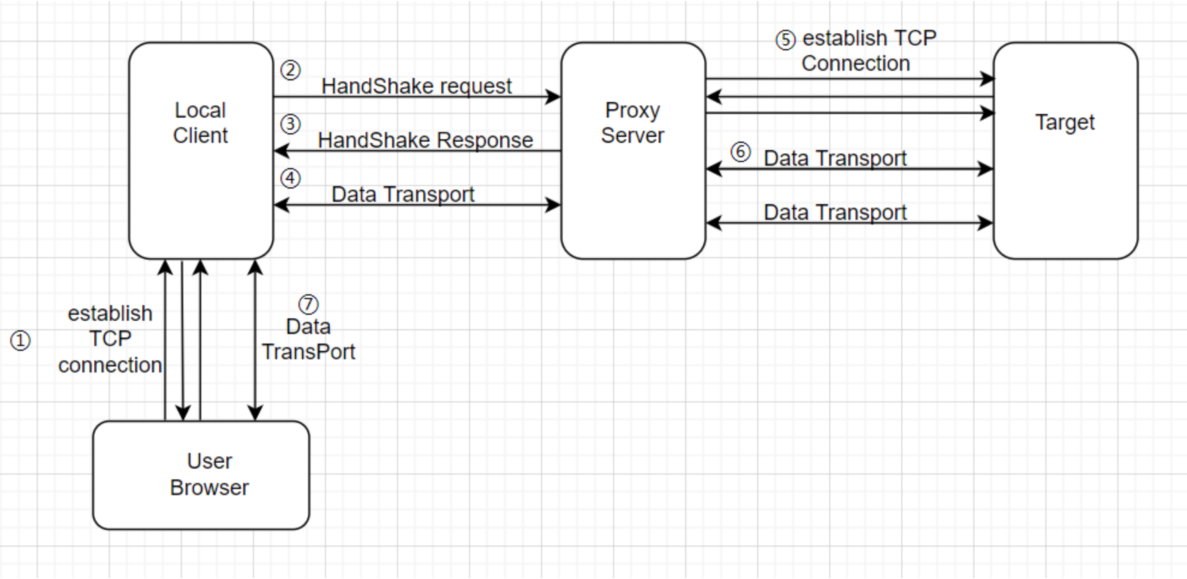
MCProxy



A simple tcp based http proxy, with some security features.

一个安全的HTTP代理工具, 基于TCP并且实现了双向认证和流量加密等功能.

Build

1. download go dependencies
2. go build -o MCProxy.exe .\app\main\MCProxy.go
3. GOOS=linux GOARCH=amd64 go build -o MCProxy_linux_amd64 .\app\main\MCProxy.go

Usage

1. config client.json and server.json
   - config ClientPK,ServerPK,ClientSK in client.json
   - config ClientPK,ServerPK,ServerSK,clients in server.json
2. run server:

```
usage: MCProxy server [<port>] [<configPath>]

server mode

Flags:
  --help  Show context-sensitive help (also try --help-long and --help-man).

Args:
  [<port>]        port to accept client's connection
  [<configPath>]  config path, default ./server.json
```

for example: ./MCProxy server 4321 ./server.json

3. run client:

```
usage: MCProxy client <port> [<server>] [<configPath>]

client mode

Flags:
  --help  Show context-sensitive help (also try --help-long and --help-man).

Args:
  <port>          client local port
  [<server>]      server address:port. (like 192.168.6.131:1234)
  [<configPath>]  config path, default ./client.json
```

for example: ./MCProxy client 1234 127.0.0.1:4321 ./client.json

4. set your browser's proxy to localhost:1234

Protocol

Packets

1. HandShakeRequest

```
type HandShakeRequest struct {
    MsgType byte        // const 1
    ClientID []byte     // 20 byte, Sha1 Of Client' PublicKey
    TimeStamp uint64
    Nonce uint32
    SPk []byte          // 68 byte,  Client's Session PK
    HashCode []byte     // 20 byte, Sha1(MsgType || ClientID ||
TimeStamp || Nonce || SPk || Sha1(Server's PublicKey || Client's
PublicKey))
    Signature []byte    // ECDSA_SIGN(HASHCODE)
}
```

2. HandShakeResponse

```
type HandShakeResponse struct {
    MsgType byte        // const 2
    TimeStamp uint64
    Nonce uint32        // request's nonce + 1
    SPk []byte          // 68 size , Server's Session PK
    HashCode []byte     // 20 size, Sha1(MsgType || TimeStamp ||
Nonce || SPk || Sha1(Server's PublicKey || Client's PublicKey))
    Signature []byte    // ECDSA_SIGN(HashCode)
}
```

3. DataTransport

```
type DataTransport struct {
    MsgType byte            // const 3
    Counter byte
    Timestamp uint64
    HashCode []byte         // 16 byte HMAC(SessionKey,(MsgType
|| Counter || Data))
    Data []byte
}
```

HandShake

client send HandShakeRequest:

1. calculate ClientID : sha1(ClientPKBase64Str)
2. generate timestamp1,random nonce1
3. generate a pair of sessionPK1 and sessionSK1, used for ECDH key-exchange(curve P256).
4. use sessionPK1 to fill HandShakeRequest
5. calculate HashCode : Sha1(MsgType || ClientID || TimeStamp || Nonce || SPk || Sha1(Server's PublicKey || Client's PublicKey))
6. use ClientPrivateKey to sign with ecdsa.
7. send HandShakeRequest to server

server received HandShakeRequest from client:

1. search client's PublicKey via clientID
2. verify timestamp, valid within three seconds
3. verify hashCode and signature
4. verify finished

server send HandShakeResponse:

1. generate timestamp2
2. calculate nonce2 = nonce2 + 1
3. generate a pair of sessionPK2 and sessionSK2.
4. calculate HashCode : Sha1(MsgType || TimeStamp || Nonce || SPk || Sha1(Server's PublicKey || Client's PublicKey))
5. use ServerPrivateKey to sign.
6. send HandShakeResponse to server

client received HandShakeResponse:

1. verify timestamp, valid within three seconds
2. verify nonce, hashcode and signature
3. verify finished

SessionKey:

- client: p256.ComputeSecret(sessionSK1,sessionPK2)
- server: p256.ComputeSecret(sessionSK2,sessionPK1)

Data Transfer

1. Counter:
    1. initial value = 0
    2. Counter = Counter + 1 after send or receive a data packet
2. send data:
    1. fill counter and generated timestamp
    2. calculate HMAC Code: HMAC(SessionKey,(MsgType || Counter || Data))
    3. fill data and send packet
    4. counter = counter + 1
3. receive data:
    1. verify counter. localCounter = packetCounter
    2. verify timestamp, valid within three seconds
    3. verify HMAC
    4. verify finished
    5. counter = counter + 1

速度测试

与wireguard对比, 下面是使用代理访问目标主机10次的测速,单位秒.

1. wireguard:

```
1.2388103008270264
1.516993761062622
1.7237181663513184
1.50636625289917
1.6959075927734375
1.2456104755401611
1.328458309173584
1.2895402908325195
1.2874672412872314
1.226855754852295
```

2. MCProxy:

```
2.0181310176849365
1.7600150108337402
1.9579222202301025
1.9202182292938232

1.7823388576507568

1.8379218578338623
1.9282798767089844
1.8139033317565918
2.089669942855835
1.8999838829040527
```

TODO

- □ 优化输出日志
- ☑ 解决访问速度慢的问题 （网络连接未及时释放导致的问题）