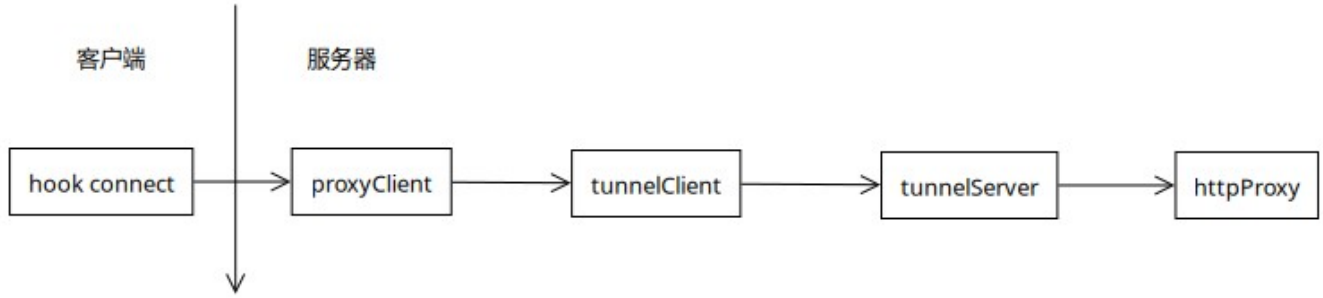
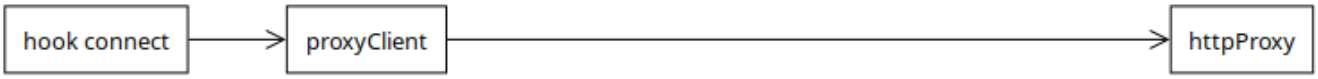
一，调研阶段的 5 个模块



hook connect， 自己编码实现
proxyClient， 自己编码实现
tunnelClient, tunnelServer，使用 gmsm 库（https://github.com/tjfoc/gmsm），自己编码实现
httpProxy，使用 squid

二，调研流程

1, 直接 hook connect,将目标地址改为 httpProxy 地址，流量不通



2, 分析国能，国电安全隧道
Android 隧道分两类实现，
a,webview 开启 http 代理，各个 android 版本方法不一
b,okhttp 等非 webview,hook Socket 类修改 connect, 注意 webview 不走 java 层 Socket，所以 hook java 层的 connect 并不通用
全平台通用的方案是 hook libc 的 connect，android,linux,windows,ios 都一样

需要实现一个类似 webview 开启 http 代理后的功能，对应的就是 proxyClient 模块
proxyClient 支持将原生 http, https 流量转为代理客户端流量
http 协议直接修改 GET 请求
https 协议增加 CONNECT 请求

下面是 hook connect 代码

```
void send_target_host(int sockfd, char* orignIp){
    char targetHost[128]={0};
    snprintf(targetHost, 128, "TARGET_HOST:%s\n", orignIp);
    int ret = send(sockfd, targetHost, strlen(targetHost), 0);
    if(-1 == ret){
        __android_log_print(ANDROID_LOG_DEBUG, "native-lib", "send errno=%d, %s",
errno, strerror(errno));
    }
}
int new_connect(int sockfd, const struct sockaddr *addr,
            socklen_t addrlen){
    struct sockaddr_in server;
    char orignIp[64];
    char newIp[64];
```

```
    if(addr->sa_family == AF_INET){
        server.sin_family = AF_INET;
        server.sin_port = htons(targetPort);
        server.sin_addr.s_addr = inet_addr(targetIP);

        sa_data_2_str((char*)addr->sa_data, (char*)orignIp, 64);
        sa_data_2_str((char*)((struct sockaddr *)&server)->sa_data, (char*)newIp,
64);

        int ret = old_connect(sockfd, (struct sockaddr *)&server, addrlen);

        if(0 == ret){
            send_target_host(sockfd, orignIp);
        }else{
            if(errno == EINPROGRESS){
                fd_set wset;
                FD_ZERO(&wset);
                FD_SET( sockfd, &wset);
                select(sockfd+1, NULL, &wset, NULL,NULL);
                send_target_host(sockfd, orignIp);
            }
        }

        return ret;
    }

    return old_connect(sockfd, addr, addrlen);
}
```

3, 增加加密隧道

使用 github 的开源项目 gmsm

使用国密证书配置，隧道自动采用 GMTLS 协议，

使用标准证书配置，隧道自动采用 TLS 协议，

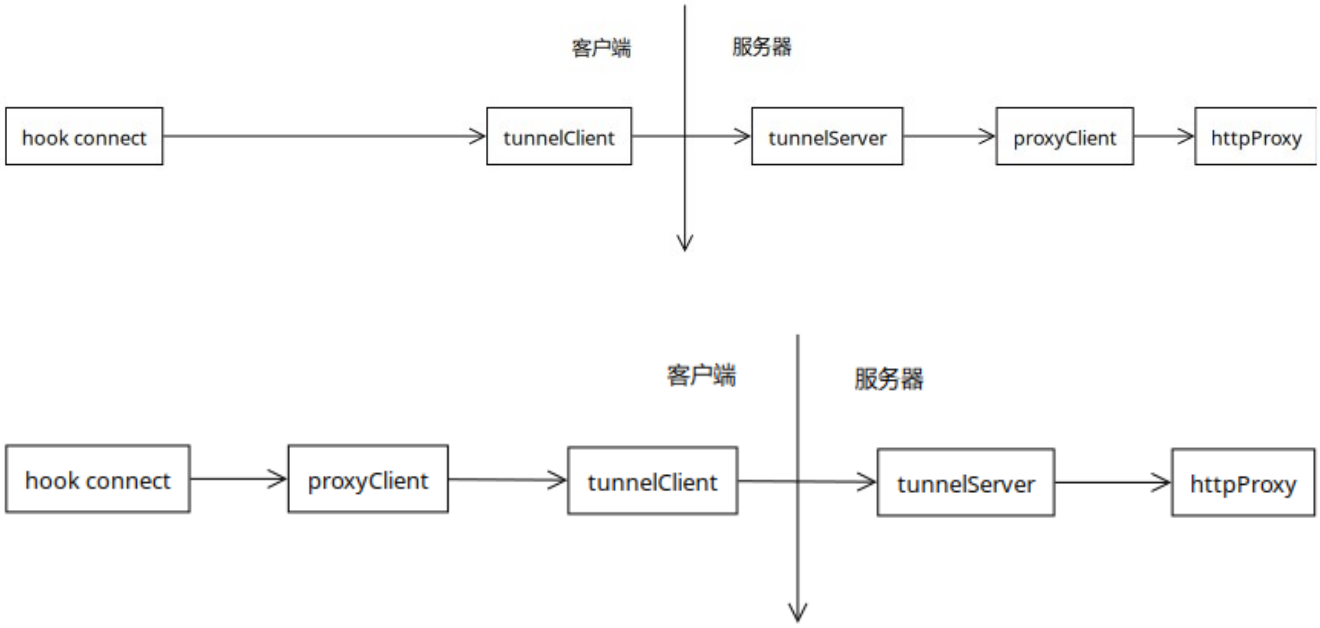
证书的时效性是在生成证书的时候写入的，使用自己的根证书 root.cer 对服务器和客户端进行签名

下面是客户端代码

```
func StartSTLClient(src string, dest string) {
        log.SetFlags(log.LstdFlags|log.Lshortfile)
        l, err := net.Listen("tcp", src)
        if err != nil {
                log.Panic(err)
        }

    initCer()
        for {
                client, err := l.Accept()
                if err != nil {
```

```
                log.Panic(err)
            }

            go handleClientRequestClient(dest, client)
        }
}


var config *gmtls.Config

func initCer(){
        // 信任的根证书
        certPool := x509.NewCertPool()
        cacert, err := ioutil.ReadFile("root.cer")
        if err != nil {
            log.Fatal(err)
        }
        certPool.AppendCertsFromPEM(cacert)
        cert, err := gmtls.LoadX509KeyPair("sm2_cli.cer", "sm2_cli.pem")

        config = &gmtls.Config{
            GMSupport:    &gmtls.GMSupport{},
            RootCAs:      certPool,
            Certificates: []gmtls.Certificate{cert},
        }
}

func handleClientRequestClient(dest string, client net.Conn) {
        server, err := gmtls.Dial("tcp", "localhost:50052", config)
        if err != nil {
            panic(err)
        }
        defer server.Close()

        go io.Copy(server, client)
        io.Copy(client, server)
}
```

三，实际开发的模块划分



区别在于 proxyClient 的位置，位于服务端后期的维护升级比较方便

后期可 hook dns 请求, 与 hook connect 配合，实现仅对白名单流量进行重定向以及支持自定义域名