网络安全工程与实现 HW4 设计文档

刘晓义 2017011426, 任乐图

2022 年 12 月 11 日

目录

1 协议设计

设计为一个 4 层代理，传递 TCP / UDP 协议的内容。

1.1 攻击模型

在我们考虑的攻击模型中，假设攻击者具有以下能力/信息：

- 可能位于链路中间人上。
- 拥有小于 2022 年地球上全部计算资源总和以内的计算能力。
- 可能拥有用户权限。在这种情况下，需要防止的攻击是针对其他用户的。特别地，这意味着握手包不能使用 Pre-shared key + 对称加密。

假设攻击者**没有**以下能力/信息

- 我们假设攻击者无法获得服务器和客户端进程本身的任何信息。这包括在主机上测量的执行时间、随机数源等。原因是我们不太懂 Time-invariant cryptography（笑）。
- 对于上游链路：
  - 在考虑攻击者破解消息内容，或者根据侧信道阻断（例如发送 Reset）的时候，我们假设攻击者不控制下游服务器，或无法监听下游链路。否则如果我们希望保持低延迟，攻击者在时间序列上可以很容易推断相关性。
  - 在考虑攻击者对消息篡改的能力时，我们假设攻击者在下游链路上只能有监听能力，但无法篡改。否则服务器就无法保证支持所有的连接（例如明文 HTTP）。
  - 特别地，在考虑攻击者尝试破解密钥时，我们**允许**攻击者在下游依旧有监听、篡改消息的能力。因此我们需要选择保证 CCA2-resistent 的 Cipher。

1.2 目标

我们希望能够达成的保护包括：

- 保证信息的保密性（Confidentiality），权威性（Authenticity）和完整性（Integrity）。更细节的：
  - 我们需要识别对消息的篡改，重放
  - 我们需要保证攻击者无法得到消息内容
  - 我们需要保证攻击者无法获取（自己之前未掌握的）密钥，包括 Session key 和 User key。
  - 攻击者无法识别用户
- 在可用性上，尽量减小协议被识别的可能，并在被阻断之后可以在对上层连接透明地情况下重连。但在攻击者持有用户帐号的时候，我们**允许**攻击者对于服务器端口的识别，以及根据这一信息对于客户端连接的识别。

为了简化客户端的实现，客户端启动的时候指定一个代理出口连接的 IP 、四层协议和端口，如果想连接到其他地方需要再启动新的客户端，类似 socat 的工作方法。这只是为了方便客户端的实现，使得我们不用设计上游应用（curl，浏览器）和客户端之间的协议。**服务端和协议设计并无这一限制**。

为了实现方便，我们不考虑的东西包括：

- 细节的性能考量。我们希望协议能有最小的 Overhead，但有些时候为了设计简单，我们会传递较多的包和元信息。
- 连接级别的向前安全性。我们不会在一个连接内修改 Session key。当然，不同连接还是会使用不同的 Session key。
- 对于 ECDHE 过程的识别。如果需要伪装这部分连接，可能最好的办法是完全实现一个内容随机的 TLS ClientHello。这本身不是技术上的困难，只是实现很麻烦，所以我们忽略了这一部分。在 ECDHE 之后，剩余的握手和消息传递协议设计都会对侧信道识别有所考虑。

  - 当然，如果 TLS 的 ECH 扩展标准化了，最好的办法是直接用 TLS 就好了。可惜现在还没有，OpenSSL 也并未合并这部分修改，所以只能自己糊协议了（

- 在重连时，不保证没有消息丢失。这部分需要上层协议保证（例如使用 HTTP/3）。这是为了减少协议设计的麻烦，不用设计 Ack 机制。当然，一个简单的方法是切换成 3 层代理，并没有协议设计上和技术上的限制，只有连接的时候地址格式发生变化。但是在实现上这需要使用 Raw socket 实现，并且客户端需要自己带一个 TCP 栈，太麻烦了。

2 协议

下层协议是 TCP。协议包含两部分，握手和一般消息传递。

2.1 准备

首先，双方持有以下内容：

服务器：拥有一个 ECDSA (NIST P256) 公钥密钥对。客户端持有公钥。这是用来验证服务器的。**攻击者允许持有公钥**

对于每个用户，拥有一个 UID <-> 密码对。UID 是 UUID (32 bytes)，密码应为 ECDSA (NIST P256) 公密钥对。服务器持有公钥，客户端持有公密钥。

2.2 握手

双向 TCP 建立完成后，首先传递的消息永远是 ECDHE (也是 NIST P256) 握手。ECDHE 协商出的密钥作为本次链接的 Session key，并一直用到连接结束。

之后传递的任意消息都使用 Session key 加密，Cipher 使用 chacha20poly1305[1]，并可以认为是在 TCP 流之上的 Datagram 协议[2]，具有以下格式：

```
| Length Nonce (12 bytes) | Length + Tag (uint32_t, 4 + 16 bytes) | Message Nonce (12 bytes)
```

- Msg Length = 0 时，这是一个控制包，Message 部分包含一个 uint64_t。
  - Message = 0: 正常关闭连接
  - Message = 1: Reset 连接
- Msg Length > 0 是，这是正常的握手或者数据包。

为了防止重放，对 Nonce 添加 Bloom filter。

剩余的握手消息包括：

- 服务器发送一个对于自己的 ECDHE 发送的随机数的签名，之后附带垃圾，长度应为 64 byte - 256 byte 随机。
- 客户端进行验证。如果验证通过，客户端发送以下结构：

[1] 流密码可以省去 Padding

[2] 使用 TCP 是为了方便保证 Datagram 大小没有限制，并且不用考虑重排

```
| OpCode (1 byte) | Reconnect Token (32 bytes) | Addr | UID (16 bytes) | Sig | Garbage |
```

其中:

- OpCode 拥有以下含义:
  - 0: 新建连接，提供 Reconnect Token 方便重连。Reconnect Token 对于同一个 (UID, Addr) tuple 不允许重复。
  - 1: 继续之前的连接，使用 Reconnect Token 进行识别。重连时需要 Addr 和启动的时候相同。
- Addr 是下游服务器地址，拥有以下格式:

```
| Types (1 byte) | IP (4 or 16 bytes) | Port (2 bytes) |
```

  - Types 的高半字节表示是 IPv4 (0) 还是 IPv6(1)，低半字节表示是 TCP (0) 还是 UDP(1)。
  - IP 和 Port 是地址和端口
- UID 是用户 ID
- Sig 是对于客户端发送的 ECDHE 随机数的签名
- Garbage 是随机长的的垃圾，总消息长度应小于 2048 Bytes.

服务器如果接受，则保持连接打开，并在 OpCode = 0 的情况下打开对于目标的连接。至此握手完毕，服务器和客户端都验证了对方的权威性，并建立起了加密信道。

2.3 消息传递

消息传递部分正常进行。特别地，如果客户端-服务器连接断开，双方应该暂时保持上游、下游连接，方便重连。如果重连 3 次依旧失败，那么再断开上下游连接。目前重连的间隔是 10s。

Connection Close / Reset 会被直接映射到上游、下游连接上。当上游、下游连接完全关闭后，客户端或服务端允许直接关闭客户端-服务器连接。当客户端-服务器连接完全关闭后，来自上游、下游的任意消息应被 Drop。

3 其他设计

以下是我们考虑的其他设计:

3.1 直接使用 PSK

一个简单的设计是单向握手，用户直接使用 PSK AEAD 加密自己的握手包，这样可以省去 Challenge 过程，直接使用 AEAD 进行用户验证。缺点是要不握手包共享 PSK，要不服务器需要对于所有用户的 PSK 依次尝试解码握手包。当用户量比较大的时候，可能成为被 DoS 的缺陷。

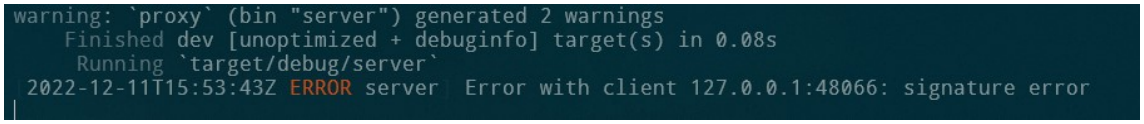

图 1: 代理服务器 Log

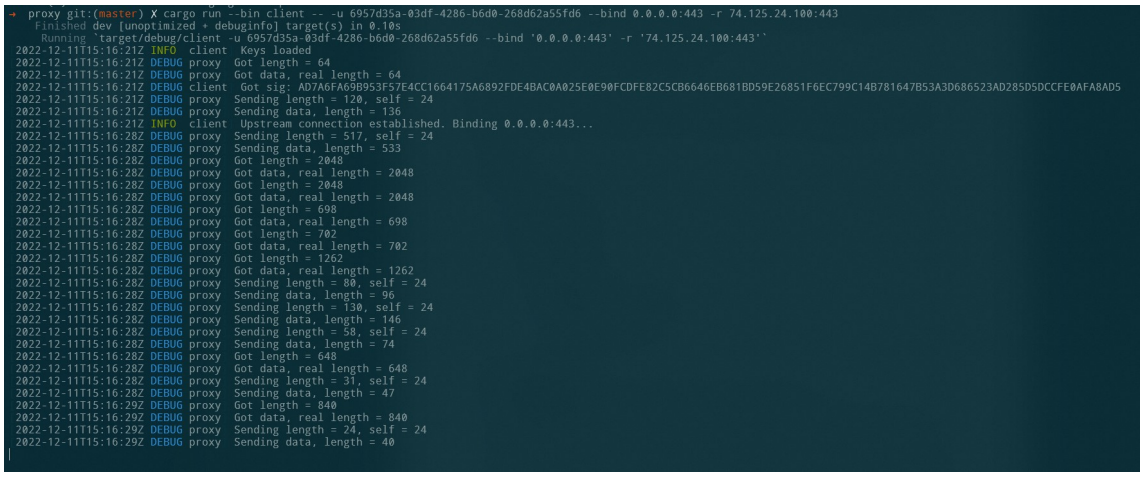

图 2: 代理客户端 Log

4 运行结果

将服务器放置在公网上，Log 见附录 B。当给出错误的客户端密钥时，Log 如图 1

A 访问 Google 日志

客户端的 Log 见图 2

curl log 为:

```
  ~ curl --resolve google.com:443:127.0.0.1 https://google.com -v
* Added google.com:443:127.0.0.1 to DNS cache
* Hostname google.com was found in DNS cache
*   Trying 127.0.0.1:443...
* Connected to google.com (127.0.0.1) port 443 (#0)
* ALPN: offers h2
* ALPN: offers http/1.1
*  CAfile: /etc/ssl/certs/ca-certificates.crt
*  CApath: none
* TLSv1.0 (OUT), TLS header, Certificate Status (22):
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.2 (IN), TLS header, Certificate Status (22):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS header, Finished (20):
```

```
* TLSv1.2 (IN), TLS header, Supplemental data (23):
* TLSv1.3 (IN), TLS handshake, Encrypted Extensions (8):
* TLSv1.3 (IN), TLS handshake, Certificate (11):
* TLSv1.3 (IN), TLS handshake, CERT verify (15):
* TLSv1.3 (IN), TLS handshake, Finished (20):
* TLSv1.2 (OUT), TLS header, Finished (20):
* TLSv1.3 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.2 (OUT), TLS header, Supplemental data (23):
* TLSv1.3 (OUT), TLS handshake, Finished (20):
* SSL connection using TLSv1.3 / TLS_AES_256_GCM_SHA384
* ALPN: server accepted h2
* Server certificate:
*  subject: CN=*.google.com
*  start date: Nov  7 08:17:21 2022 GMT
*  expire date: Jan 30 08:17:20 2023 GMT
*  subjectAltName: host "google.com" matched cert's "google.com"
*  issuer: C=US; O=Google Trust Services LLC; CN=GTS CA 1C3
*  SSL certificate verify ok.
* Using HTTP2, server supports multiplexing
* Copying HTTP/2 data in stream buffer to connection buffer after upgrade: len=0
* TLSv1.2 (OUT), TLS header, Supplemental data (23):
* TLSv1.2 (OUT), TLS header, Supplemental data (23):
* TLSv1.2 (OUT), TLS header, Supplemental data (23):
* h2h3 [:method: GET]
* h2h3 [:path: /]
* h2h3 [:scheme: https]
* h2h3 [:authority: google.com]
* h2h3 [user-agent: curl/7.86.0]
* h2h3 [accept: */*]
* Using Stream ID: 1 (easy handle 0x5593b6316e60)
* TLSv1.2 (OUT), TLS header, Supplemental data (23):
> GET / HTTP/2
> Host: google.com
> user-agent: curl/7.86.0
> accept: */*
>
* TLSv1.2 (IN), TLS header, Supplemental data (23):
* TLSv1.3 (IN), TLS handshake, Newsession Ticket (4):
* TLSv1.3 (IN), TLS handshake, Newsession Ticket (4):
* old SSL session ID is stale, removing
* TLSv1.2 (IN), TLS header, Supplemental data (23):
```

```
* TLSv1.2 (OUT), TLS header, Supplemental data (23):
* TLSv1.2 (IN), TLS header, Supplemental data (23):
* TLSv1.2 (IN), TLS header, Supplemental data (23):
< HTTP/2 301
< location: https://www.google.com/
< content-type: text/html; charset=UTF-8
< cross-origin-opener-policy-report-only: same-origin-allow-popups; report-to="gws"
< report-to: {"group":"gws","max_age":2592000,"endpoints":[{"url":"https://csp.withgoogle.com
< date: Sun, 11 Dec 2022 15:17:37 GMT
< expires: Tue, 10 Jan 2023 15:17:37 GMT
< cache-control: public, max-age=2592000
< server: gws
< content-length: 220
< x-xss-protection: 0
< x-frame-options: SAMEORIGIN
< alt-svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000,h3-Q050=":443"; ma=2592000,h3-Q046=
<
* TLSv1.2 (IN), TLS header, Supplemental data (23):
<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="https://www.google.com/">here</A>.
</BODY></HTML>
* TLSv1.2 (IN), TLS header, Supplemental data (23):
* Connection #0 to host google.com left intact
```

B 访问 Google 日志