第27卷 第3期 石家庄铁道大学学报(自然科学版) Vol. 27 No. 3
2014年09月 JOURNAL OF SHIJIAZHUANG TIEDAO UNIVERSITY (NATURAL SCIENCE) Sep. 2014

基于优先级策略的TLS握手协议研究

高志伟， 耿金阳

(石家庄铁道大学 信息科学与技术学院,河北 石家庄 050043)

摘要: 当大量的用户通过TLS协议访问服务器时,会带来较大的负担,同时浏览器进程关闭时会清除用户所保存下来的session信息,当再次连接时,又需要完整的握手过程。对TLS协议进行分析,在经典握手协议的基础上提出了服务器端session的缓存机制的改进,提高了服务器的缓存利用率;并在客户端实现了运用cookie进行会话重用。结合Java相关技术,对其TLS环境进行配置,并在其上进行了服务器端与客户端的实验,然后对经典TLS握手协议进行效率测试。

关键词: TLS; session; 会话重用

中图分类号: TP311 **文献标志码:** A **文章编号:** 2095-0373(2014)03-0069-06

0 引言

IE浏览器和服务器在传输的过程中使用TLS(Transport layer Security)协议来保证敏感数据的安全性并且验证数据的准确性,但是TLS协议也经常为客户端和服务器端带来困扰。TLS中不正确的配置与证书警告是非常常见的,他们可以导致数据的泄露并且可能会引起其他的问题。对于TLS协议使用者来说,可能会面临导致"会话欺骗"(session hijacking)的情况,并且TLS也带来了相对较重的负担。通过TLS建立连接的服务器站点当然也会使用户感到延迟。此外,占用用户的资源,即使是很小的额外的用户延迟,也会给网站带来网络拥堵,利用率降低,额外的开销等[1]。

在用户和服务器开始发送数据时,标准的TLS握手协议中需要通过两次验证性对话,这将使网络延时增加,进而影响用户体验并影响到TLS的应用与推广。此外,IE浏览器还必须通过证书废除协议来验证服务证书的有效性(比如:在线证书状态查询协议OCSP)[2],这就又增加了等待时间。因此减少TLS握手等待时间是很关键的,这会使TLS得到更广泛的使用并加强网络的安全性。

国际上很多提案都通过减少握手验证数据传输回合的办法达到了提高TLS性能的作用。较有代表性的如:Fast-track通过客户端缓存长效性参数减少了一个握手环节[3];TLS False Start通过提前记录安全算法套件信息减去一个回合[4];最有效的Snap Start方案通过客户端提前完成与服务器的完整握手并缓存静态参数数据,使再次连接握手时免去了握手环节[5];以及2011年由Lin-Shung Huang、Emily Stark、Dinesh Israni、Collin Jackson等人提出的perfetching and pervalidating提前取得证书及验证、TLS snap start、会话重用等机制[6]。RFC4507文档中提出安全使传输层(TLS)服务器恢复会话不需要让客户端会话保持连接状态的机制。TLS服务器封装会话状态变成"票据"并将其转发给客户端。客户端可以随后使用所获得的"票据"恢复会话[7]。

但是不论哪种方案都有各种各样的附加代价,比如Fast-track、False需要客户端提供缓存空间才能减少连接的时间,RFC4507中提出的机制需要大量占用服务器端的内存才能保持连接状态,而且当大量的用户通过TLS协议访问服务器时,会带来较大的负担,以及浏览器进程关闭时会清除用户所保存下来的

DOI: 10.13319/j.cnki.sjztddxxbzrb.2014.03.15
收稿日期: 2013-06-06
作者简介: 高志伟 男 1972年出生 副教授

session 信息,当再次连接时,又需要完整的握手过程。这些问题都有待技术人员的进一步解决。

现从经典握手协议过程入手,模拟会话重用过程,并根据现在研究现状,分别从服务器端提出缓存机制的改进,客户端提出运用 cookie 进行会话重用的实现,对于服务器来说,逻辑上扩充了其缓存,并且提高了缓存利用率;客户端来说,减少了再次进行完整握手过程的时间,提高了 TLS 在实际中使用的效率。

1 TLS 经典握手协议分析

完整 TLS 握手过程:完整的握手过程是之前没有通信过的双方建立 TLS 的过程,该过程也对绝大多数情况适用。图 1 给出了完整的握手过程消息流[6]。

TLS 是用于在客户端和服务器端之间对其通信进行加密和认证的协议。为了建立起一个安全可靠的连接,客户端和服务器端都必须对对方身份进行认证(通过有 Certificate Authority 发布的证书来认证),然后才可以进行握手过程。利用在握手过程商量好的 ciper suite,客户端和服务器端商量出密钥用以保证握手过程后的数据通信的安全。

在 web 通信过程中,TLS 保证了 web 服务器和 web 站点之间的 http 通信的隐私性和数据保密性。图 1 显示了一个完整的 TLS 握手过程,其中用了 RSA 加密算法,并且没有客户端的证书验证。ClientHello 和 ServerHello 用于客户端和服务器端之间进行信息交换从而达成一个约定,约定的内容包括:要用哪个版本的 TLS 以及用哪个 cipter suite。这些初始信息也允许客户端和服务器端进行随机数(从 session key 提取出来)的交换。客户端的随机数包括客户端的时间信息。当服务器端接收到 ClientKeyExchange 消息后,双方都可以直接知道主密钥(master key),主密钥接下来是用来加密数据的。ChangeCiperSpec 消息表示另一方接下来发送的信息将会用商量好的 ciper suite 来加密。Finish 信息包含着整个握手过程的哈希值,用来保证双方在握手过程没有被黑客攻击篡改信息。经历二轮的来回通信,并且接收到加密的 http 的请求,客户端才会发送第一个数据信息。交换的过程如图 2 所示。

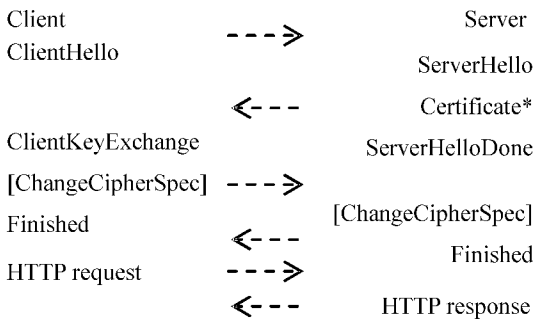

图 1 标准 TLS 握手过程(用 RSA 密钥交换但没有客户端的证书认证)

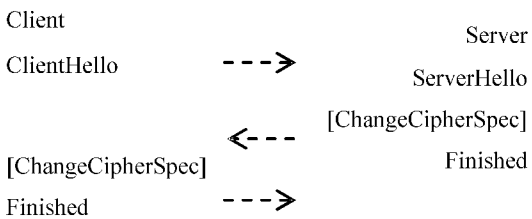

图 2 TLS 握手会话重用过程

TLS 连接也允许恢复以前的会话。如果以前的会话(session)将要被恢复,服务器端会在 ServerHello 信息中提供一个会话 id。为了恢复一个会话,客户端首先要有以前保存的 session id 并且在恢复会话握手过程中将其放在 ClientHello 信息中。

服务器端可以通过会话 id 来判断特殊的连接,会话 id 在 ServerHello 消息中有自己专门的字段。客户端和服务器端在双方都同意的情况可以进行会话的重用。客户端在自己发送消息中存放会话 id,表示他希望进行会话重用,服务端通过在回复的 ServerHello 消息中存放同样的会话 id,表示他接受了客户端的请求。然后客户端和服务器端直接进行 Change CipherSpec 和 Finish 消息的交换。(按照以前会话商量好的参数和密钥)。

2 服务器端缓存机制基于优先级策略的改进方案

如果 session 按照 100byte 估算[6]。假设一台服务器有 4G 内存可用于应用缓存,那么一天中该服务器最多能存储的 session 数为 $4 \times 10e8$ 个。当然,这是最坏的情况,因为正常情况中在一天中会有重用的会话。按照 1 000 次/s 握手,每个 session 为 100byte 那么服务器 Cache 会以 3 MB/min 增加[8]。

在访问量较大的服务器上这会给服务器带来更大的负担,出于这一原因,在运用时就必须要降低 Session 可缓存的时间。如亚马逊网站将 Session 缓存的有效时间设定为 2 min, mod_ssl 中也默认设定为 5 min。因此有需要对服务器的缓存机制做出一定的改变,以提高服务器应对大量客户端的访问。

本节就这一问题,利用操作系统中 LRU 置换算法对服务器缓存进行维护操作,从而提高服务器的缓存利用率。

2.1　最近最久未使用置换算法(LRU)

操作系统中,虚拟存储技术是目前普遍采用的存储管理技术,它的意义就在于从逻辑上扩充其容量。其中请求分页存储中新的页面替换内存中已有的页面,如何挑选,则需要根据置换算法来确定。本文将请求分页存储管理中置换算法应用到 TLS 握手协议中服务器端,将其置换的页面变为 session,使其逻辑上扩充缓存的容量,提高其缓存利用率。

最佳置换算法的和先进先出置换算法在页面置换算法中是两种极端的算法。最佳置换算法是一种理想化的算法,必然具有最好的性能,但在现实中并不能实现。而先进先出置换算法则是最容易实现的一种算法,但是它在性能上也是最差的,在实际中也很少使用。最近最久未使用(LRU)置换算法无论在性能上还是在实现上都介于这两种算法之间。LRU 置换算法所根据的原理是: 当页面(本文为 session)调入内存后,由于无法预测页面将来的使用情况,只能把“最近的过去”作为“最近的将来”的近似。因此,LRU 置换算法将最近最久未使用的页面(session)予以淘汰。

本文中 LRU 算法的实现,可利用一个特殊的队列来保存当前的保存了的各个会话的 id。当客户端进程进行会话重用时,该会话 id 从队列中提出,并将它放入对首。因此,队首始终是最近进行会话重用的会话 id,而队尾则是最近最久未进行会话重用的会话 id。

2.2　原包修改

JSSE(Java Secure Socket Extengsion)是 Java 中用来支持 SSL 与 TLS 安全通信的 API 包。它能够提供数据加密、证书身份验证等。在引用 JSSE 时,可以设置相关的属性,包括: 系统属性以及安全属性。它屏蔽了细节,并且调用 API 进行互相认证,还在 Socket 上的传输的数据进行加密。

由于缓存操作中需要对 session 对象、sessionId 对象进行操作,而原 Java 的 jsse.jar 包的类库中对于底层服务对象都是禁止访问的,因此需要对该包中的类进行部分修改。

此处主要修改 jsse.jar 包中的 SessionId 类、SSLSessionImpl 类中部分代码。如下:

SessionId 修改:

(1) 修改类属性为 public。

(2) 加入构造函数。

```
public SessionId( byte abyte0 [])
        { sessionId = abyte0;
        }
```

(3) 加入 ID 设置方法。

```
public void SetSessionId( byte abyte0 [])
{
sessionId = abyte0;
}
```

SSLSessionImpl 修改: 修改 getSessionId 属性为 public。

由于 session 缓存记录位于 jsse.jar 包的 final 类 SSLSessionContextImpl 中,从外部无权限对类中的 cache 对象进行操作,在权衡对封装安全性的考虑之后采用 Java 反射机制获取 SessionCache、以及 SessionHostPortCache 对象属性,对服务器缓存进行操作。

2.3　优先调度算法的实现

本节对服务端的 cache 中对 session 进行优先级的设置,当最近调用该 session 后,该 session 的优先级

提高。所以,从 cache 中最先删除的肯定是最近这段时间没有调用过的 session(即:操作系统中最近最久未使用算法(LRU))。流程图见图3。

具体操作如下:

(1)从 Cache 对象获取所需删除的低优先级 session 将其从 SessionCache、SessionHostPortCache 中删除,以减少服务器缓存。

(2) byte ‖ id = sslSock. getSession( ) . getId( ) ;这是用来判断是否已记录标志,若没有则添加如列表,若有则将其删除并重新加该入队列,提高其优先级。

(3)用 count 记录缓存 session 数量,当超过某一值时创建 MyCache 实例清理优先级低的几个 session 缓存。

(4)服务器线程中加入一个集合列表用于存储缓存中对应的 session 的 ID,并以优先调度算法对缓存进行控制。

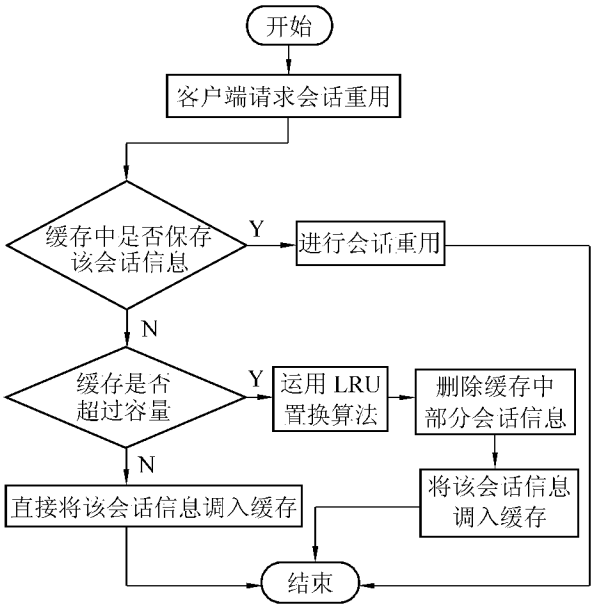

图3 优先调度算法流程图

3 利用客户存储实现会话重用

虽然在 TLS 安全连接时可以进行会话重用,但是服务器与浏览器的会话重用存在问题,当浏览器进程关闭时都会清除前边用户所保存下来的 Session 信息,当再次启动浏览器进程,连接 HTTPS 服务器端时又需要再次进行完整的握手连接。Session resume 机制也就没有发挥它的作用,严重的影响了 HTTPS 安全连接方式的使用与发展。所以本小节对此提出问题,并在 Java 平台进行模拟实验,下边对 Java 平台中的类进行分析。

3.1 SSLsession 分析

SSLSession 是 Java 平台中进行会话重用重要的类,SSLSession 由 SSLSessionImpl 类实现,其内容包括 SSL 连接的可缓存内容 sessionContext 参数、creationTime、lastUsedTime 等。其中证书链及证书参数只是对对象的引用并非 SSLsession 中含有证书实体。

在 JSSE 类库中,服务器在底层代码中建立 ServerContext,负责每次连接时 SSLSession 的存取操作。当完整握手验证完成时将 SSLSession 存入设定好的 Cache 对象中,并完成标志符 inValid、isTimeOut 的维护。如果客户端请求进行会话重用,服务器则负责验证 SSLSession 的有效性,若有效则从 Cache 对象中取出 SSLSession 读取其中的信息用于开始连接握手。每个 SSLSession 的默认有效时间为 0X15180 就是 24 h。

在客户端,浏览器在完成一次完整 TLS 连接握手之后会将 sessionId 以 Cookie 的形式缓存下来。如果浏览器在不重启的前提下就能使用该缓存进程会话重用。

3.2 实现 SSLsession 长久可用

在正常 HTTPS 连接时,浏览器会在 TLS 完整握手完成时对 sessionId 进行缓存,由此直到浏览器进程结束这段时间内都能够实现 session resume。为了广泛的实现 session resume 在对安全要求不是特别严格的环境(比如可排除 session 缓存数据被盗取使用的可能性)下可以对其在 Java 平台下进行实现:

由服务器端维护一个 sessionId 表,记录每次完成完整 TLS 握手时的客户方信息,再有新连接请求时遍历 sessionId 表读取 sessionId 用于创建 ClientHello 进行握手。

由客户端程序维持一个全局变量 hostName 用于记录当前客户名。在 SSLSessionContextImpl 类中完成对 session 的缓存及读取操作,以 hostName 作为 Key 保存 sessionId。然后在 handshaker 类中对内部类 clientHello 的构造函数进行修改,使其在建立连接请求前查找中 sessionId 记录。

3.3 Java 证书库

客户端会话重用是用 Java 模拟的,而 Java 的证书库与浏览器有一定的区别。相对于 Java 有自己的证书体系,在试验中需要将各个证书提取封装成为 KeyStore 对象结构来使用(如图 4) 。为了结合其他实验项目,需要将 pfx 格式证书或 cer 转换成为统一的 Java 中 KeyStore 标准的证书文件。

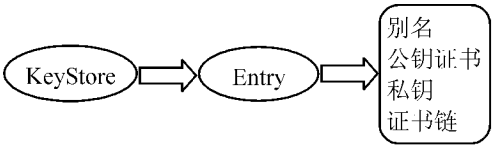

图 4　KeyStore 结构

首先需要将一个证书中的 Certificate 对象及密钥对象提取出来,然后再用 KeyStore 结构将 Certificate 对象、私钥、证书链都设定在一个证书实例中获取 KeyStore。

3.4　Java 平台下的会话重用

支持 TLS/SSL 协议的浏览器都能在启用 Cookie 和浏览器进程不重启的前提下与 HTTPS 服务器借助 Cookie 缓存完成会话重用。本实验已通过实验 Firefox 与中国工商银行服务器进行连接,分析了其过程中传输的 HTTP 报文,了解清楚该浏览器与服务器完成会话重用的实现过程,并用 Java 程序模拟发送数据,与服务器实现会话重用。

在 Java 模拟实验中,https 连接过程,客户端在第一次完成验证之后读取服务器响应信息中的 Set - Cookie 字段并保存。在第二次连接时读取该数据,写入到 http 的 GET 请求中,发送到服务器端,完成有效会话的重用。

4　实验测试

4.1　安装环境

(1) 配置 org. bouncycastle 算法库。在 NetBeans 的项目中,在项目“库”文件中增加 bcmail-jdk16-1. 44. jar 与 bcprov-jdk16-145. jar 文件。

(2) 在使用的过程中,因受到出口限制,密钥长度不能满足实验的需求。这是因为 sun 发布的权限文件做了相应限制,我们可以在 sun 的官方网站找到替换文件来减少相关限制。

(3) BC provider 添加: 找到 Java 根目录下\Java\jdk1. 7. 0 \jre\lib\security 中的 Java. security 用记事本打开,在 security. provider 处的末尾添加 security. provider. 8 = org. bouncycastle. jce. provider. BouncyCastleProvider。

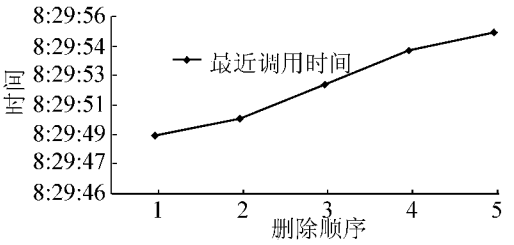

图 5　服务器缓存对象调度算法执行情况

4.2　实验结果

将 LRU 置换算法对其服务端进行改进之后,其服务器端删除的 session 的顺序如图 5 所示。

实验结果: 将 LRU 置换算法加入到服务器端后,session 从缓存中被删除的时间,是严格按照最近被调用的时间递增的顺序来进行删除的,因为要删除的肯定是最近未被调用的,即: 最近调用时间最早的一个,然后依次删除其他最近调用时间较早的 session,所以实验结果是严格按照最近被调用时间递增是完全符合逻辑,也是符合 LRU 置换算法的表现。

利用客户存储实现会话重用的实验结果如表 1 所示。

表 1　实验连接耗时　　ms

实验组	连接次数	第一次耗时	中位数	会话重用平均耗时	最长耗时
S1	5	2 110	2 110	1 664	3 428
S2	15	1 830	1 070	2 168	8 380
S3	30	1 485	469	694	3 984
S4	50	1 550	453	600	3 297
S5	100	1 608	437	545	3 801

表 1 中分了 5 组进行实验,分别为连接次数为 5,15,30,50,100 次,表中数据可看出连接中第一次的时间正常情况下比后边会话重用平均耗时的时间要长,其中第二组实验会话重用平均耗时要比第一次耗时要多,这是由于两方面原因造成,第一是由网络不稳定原因造成的,其中最长耗时为 8 380 ms; 第二是由

于进行的连接次数较少,造成数据并不是很稳定,但如第1、2、4、5组都是正常的。而且连接次数越多,中位数却趋于稳定,并且是要比第一次耗时要少的,对于突发的网络不稳定原因造成的延迟,中位数比会话重用平均耗时可能更体现会话重用的实际效果,除了第一组实验中中位数都比第一次连接耗时相等外,其他组实验都是比第一次要少。

5 结语

TLS握手协议在连接过程所带来的用户延迟,使网站和用户不愿意使用TLS。缓存后的握手协议可以去除证书的验证所用的时间,减少延迟。本文结合Java安全措施的相关内容,分别在服务器端提出缓存机制的改进,提高了服务器的缓存利用率;并在客户端实现了运用cookie进行会话重用,减少TLS握手过程中所需要的时间。

参 考 文 献

[1] Souders S. WPO-Web performance optimization [EB/OL]. 2010-5-7 [2013-5-1]. http://www.stevesouders.com/blog/2010/05/07/wpo-web-performance-optimization/.

[2] Myers M, Ankney R, Malpani A. X.509 Internet public key infrastructure online certicate status protocol - OCSP [EB/OL]. 1999-5-1 [2013-5-1]. http://www.ietf.org/rfc/rfc2560.txt.

[3] Shacham H, Boneh D. Fast-track session establishment for TLS [C]//Proceedings of the Network and Distributed System Security Symposium (NDSS). San Diego, California, [s.n.], 2002.

[4] Langley A, Modadugu N, Moeller B. Transport layer security (TLS) false start [EB/OL]. 2010-12-4 [2013-5-1]. http://tools.ietf.org/html/draft-bmoeller-tls-falsestart-00.

[5] Langley A. Transport Layer Security (TLS) Snap Start [EB/OL]. 2010-11-2 [2013-5-1]. https://www.imperialviolet.org/binary/draft-agl-tls-snapstart-00.html.

[6] Emily Stark, Huang Linshung, Dinesh. The case for prefetching and prevalidating TLS server certificates [C]//proceedings of Usenix Security. Washington, DC, USA, USENIX Association, 2010.

[7] Salowey J, Zhou H, Eronen P, et al. Transport layer security (TLS) session resumption without server-side state [EB/OL]. 2006-5-1 [2013-5-1]. http://wenku.baidu.com/view/6e00ae778e9951e79b892771.html.

[8] Hovav Shacham, Dan Boneh Eric, Rescorlahovav. Client side caching for TLS [J]. ACM Transactions on Information and System Security (TISSEC), 2004, 7(4): 553-575.

Priority-based Policy Research on TLS Handshake Protocol

Gao Zhiwei, Geng Jinyang

(School of Information Science and Technology, Shijiazhuang Tiedao University, Shijiazhuang 050043, China)

Abstract: When a lot of users access the server via TLS protocol, a greater burden is generated, and when the browser is closed, the session information about users will be cleared, and if connected again, it has to complete the full TLS handshake again. The TLS protocol is analyzed based on the classic handshake protocol, and an improved method of caching mechanism of the server-side session is designed which improves the server's cache utilization, and session resume is carried out by cookie in the client-side. Combined with Java related technologies, TLS environment is configured, experiments are conducted on server-side and client-side, and the efficiency of the TLS handshake protocol is tested.

Key words: TLS; session; session resuming

(责任编辑 车轩玉)