2011-06-03

**[tlslite readme摘要](http://marshan.iteye.com/blog/1069592)**

python之TLS使用

**版本**

tlslite version 0.3.8

**下载**

<http://trevp.net/tlslite/>

**安装**

sudo python setup.py install

**测试安装**

cd test

tls.py servertest localhost:4443 .

tls.py clienttest localhost:4443 .

ubuntu/python2.7:

/usr/local/bin/tls.py

/usr/local/lib/python2.7/dist-packages/tlslite/TLSConnection.py

**命令行**

'tlsdb.py' 管理共享钥和验证用户-共享钥，SRP密码数据库

'tls.py' 测试其他TLS实现

 运行SRP server:

  tlsdb.py createsrp verifierDB

  tlsdb.py add verifierDB alice abra123cadabra 1024

  tlsdb.py add verifierDB bob swordfish 2048

  sudo tls.py serversrp localhost:443 verifierDB

 连接server:

  tls.py clientsrp localhost:443 alice abra123cadabra

**基本使用步骤**

**Step 1 - 创建socket**

 记得设置超时处理

  from socket import \*

  sock = socket(AF\_INET, SOCK\_STREAM)

  sock.connect( ("www.amazon.com", 443) )

  sock.settimeout(10)  #Only on python 2.3 or greater

**Step 2 - 创建TLS连接**

  from tlslite.api import \*

  connection = TLSConnection(sock)

**Step 3 - 调用握手函数(client)**

 可以根据验证方式，使用不同的握手实现：

  connection.handshakeClientCert() #无需客户端验证(authentication)

  connection.handshakeClientCert(certChain, privateKey)

  connection.handshakeClientSRP("alice", "abra123cadabra")

  connection.handshakeClientSharedKey("alice", "PaVBVZkYqAjCQCu6UBL2xgsnZhw")

  connection.handshakeClientUnknown(srpCallback, certCallback)

 ClientCert私钥获取：

 #Load cryptoID certChain and privateKey.  Requires cryptoIDlib.

 from cryptoIDlib.CertChain import CertChain

 s = open("./test/clientCryptoIDChain.xml").read()

 certChain = CertChain()

 certChain.parse(s)

 s = open("./test/clientCryptoIDKey.xml").read()

 privateKey = parseXMLKey(s, private=True)

 #Load X.509 certChain and privateKey.

 s = open("./test/clientX509Cert.pem").read()

 x509 = X509()

 x509.parse(s)

 certChain = X509CertChain([x509])

 s = open("./test/clientX509Key.pem").read()

 privateKey = parsePEMKey(s, private=True)

 SRP and SharedKey都需要手动验证用户名和密码.

 不同的是，SRP较慢，但对低熵密码安全；shared keys快速，但只对高熵密码安全。

 一般情况下，SRP用于人类可识别记忆的密码，在使用随机码作为密码时才使用shared keys

 Unknown用于当无法知道server是否需要client authentication的时候

 两个回调函数SRP callback和certificate callback 通常接受一个包含(username, password)，(certChain,privateKey)或者的tuple

 通过HandshakeSettings 更多地控制handshake：

 settings = HandshakeSettings()

 settings.minKeySize = 2048

 settings.cipherNames = ["aes256"]

 settings.minVersion = (3,1)

 connection.handshakeClientSRP("alice", "abra123cadabra", settings=settings)

 重用session：

 connection.handshakeClientSRP("alice", "abra123cadabra")

 ……

 oldSession = connection.session

 connection2.handshakeClientSRP("alice", "abra123cadabra", session=  oldSession)

**Step 3 - 调用握手函数(server)**

 server只有一个handshake函数，但不同的验证方式参数不同

 SRP authentication要通过VerifierDB来验证数据库的密码

 verifierDB = VerifierDB("./test/verifierDB") #无参数代表内存数据库

 #打开已存在的

 verifierDB.open()

 #创建新的

 verifier = VerifierDB.makeVerifier("alice", "abra123cadabra", 2048)

 verifierDB["alice"] = verifier

 #执行握手

 connection.handshakeServer(verifierDB=verifierDB)

 shared key authentication

 sharedKeyDB = SharedKeyDB("./test/sharedkeyDB")

 sharedKeyDB.open()

 sharedKeyDB["alice"] = "PaVBVZkYqAjCQCu6UBL2xgsnZhw"

 connection.handshakeServer(sharedKeyDB=sharedKeyDB)

 a certificate and private key authentication

 connection.handshakeServer(certChain=certChain, privateKey=privateKey, reqCert=True)

 通过SessionCache重用客户端session

 sessionCache = SessionCache()

 connection.handshakeServer(verifierDB=verifierDB, sessionCache=sessionCache)

**Step 4 - 检查结果**

 如果握手结束没有出现异常，验证结果将被保存在connection's session object

 如下variables是常用的：

 connection.session.srpUsername       #string

 connection.session.sharedKeyUsername #string

 connection.session.clientCertChain   #X509CertChain or cryptoIDlib.CertChain.CertChain

 connection.session.serverCertChain   #X509CertChain or cryptoIDlib.CertChain.CertChain

 Both types of certificate chain object support the getFingerprint() function,

 X.509对象返回 the end-entity fingerprint并忽略其他证书

 CryptoID fingerprints (也就是 "cryptoIDs") 是基于根证书(root cryptoID certificate)的, 所以必须在CertChain中调用validate()确保真的和cryptoID通话.

 try:

 checker = Checker(\x509Fingerprint='e049ff930af76d43ff4c658b268786f4df1296f2')

 connection.handshakeClientCert(checker=checker)

 except TLSAuthenticationError:

 print "Authentication failure"

 错误类型略

**Step 5 - 交换数据**

 创建连接后，使用socket.SSL对象的read() and write()

 或者使用socket对象的send(), sendall(), recv(), 和makefile()

 会导致的异常：TLSLocalAlert,TLSRemoteAlert, socket.error, or TLSAbruptCloseError

**Step 6 - 关闭连接**

 调用close()

**HTTPTLSConnection对httplib的扩展**

  #没有验证

  h = HTTPTLSConnection("www.amazon.com", 443)

  h.request("GET", "")

  r = h.getresponse()

……

  #基于服务器X.509 fingerprint的验证

  h = HTTPTLSConnection("www.amazon.com", 443, x509Fingerprint="e049ff930af76d43ff4c658b268786f4df1296f2")

……

  #基于服务器 X.509 chain (需要cryptlib\_py)的验证

  h = HTTPTLSConnection("www.amazon.com", 443, x509TrustList=[verisignCert], x509CommonName="www.amazon.com")

……

  #基于服务器cryptoID的验证

  h = HTTPTLSConnection("localhost", 443, cryptoID="dmqb6.fq345.cxk6g.5fha3")

……

  #使用SRP手动验证

  h = HTTPTLSConnection("localhost", 443, username="alice", password="abra123cadabra")

……

  #使用shared key手动验证

  h = HTTPTLSConnection("localhost", 443, username="alice", sharedKey="PaVBVZkYqAjCQCu6UBL2xgsnZhw")

……

  #基于服务器cryptoID 使用SRP, \*AND\*手动验证

  h = HTTPTLSConnection("localhost", 443, username="alice", password="abra123cadabra", cryptoID="dmqb6.fq345.cxk6g.5fha3")

……

**XMLRPCTransport对xmlrpclib的扩展**

  from tlslite.api import XMLRPCTransport

  from xmlrpclib import ServerProxy

  #No authentication whatsoever

  transport = XMLRPCTransport()

  server = ServerProxy("https://localhost", transport)

  server.someFunc(2, 3)

……

  #Authenticate server based on its X.509 fingerprint

  transport = XMLRPCTransport(\

          x509Fingerprint="e049ff930af76d43ff4c658b268786f4df1296f2")

……

**POP3\_TLS对poplib的扩展**

**IMAP4\_TLS对imaplib的扩展**

  #To connect to a POP3 server over SSL and display its fingerprint:

  from tlslite.api import \*

  p = POP3\_TLS("---------.net")

  print p.sock.session.serverCertChain.getFingerprint()

……

  #To connect to an IMAP server once you know its fingerprint:

  from tlslite.api import \*

  i = IMAP4\_TLS("cyrus.andrew.cmu.edu", x509Fingerprint="00c14371227b3b677ddb9c4901e6f2aee18d3e45")

……

SMTP\_TLS对smtplib的扩展

  #To connect to an SMTP server once you know its fingerprint:

  from tlslite.api import \*

  s = SMTP\_TLS("----------.net")

  s.starttls(x509Fingerprint="7e39be84a2e3a7ad071752e3001d931bf82c32dc")

……

**SocketServer的使用**

  from SocketServer import \*

  from BaseHTTPServer import \*

  from SimpleHTTPServer import \*

  from tlslite.api import \*

  s = open("./serverX509Cert.pem").read()

  x509 = X509()

  x509.parse(s)

  certChain = X509CertChain([x509])

  s = open("./serverX509Key.pem").read()

  privateKey = parsePEMKey(s, private=True)

  sessionCache = SessionCache()

  class MyHTTPServer(ThreadingMixIn, TLSSocketServerMixIn, HTTPServer):

      def handshake(self, tlsConnection):

          try:

              tlsConnection.handshakeServer(certChain=certChain,

                                            privateKey=privateKey,

                                            sessionCache=sessionCache)

              tlsConnection.ignoreAbruptClose = True

              return True

          except TLSError, error:

              print "Handshake failure:", str(error)

              return False

  httpd = MyHTTPServer(('localhost', 443), SimpleHTTPRequestHandler)

  httpd.serve\_forever()

**TLSAsyncDispatcherMixIn.py**

  class http\_tls\_channel(TLSAsyncDispatcherMixIn, http\_server.http\_channel):

      ac\_in\_buffer\_size = 16384

      def \_\_init\_\_ (self, server, conn, addr):

          http\_server.http\_channel.\_\_init\_\_(self, server, conn, addr)

          TLSAsyncDispatcherMixIn.\_\_init\_\_(self, conn)

          self.tlsConnection.ignoreAbruptClose = True

          self.setServerHandshakeOp(certChain=certChain, privateKey=privateKey)

**Twisted协议**

  from twisted.internet.protocol import Protocol, Factory

  from twisted.internet import reactor

  from twisted.protocols.policies import WrappingFactory

  from twisted.protocols.basic import LineReceiver

  from twisted.python import log

  from twisted.python.failure import Failure

  import sys

  from tlslite.api import \*

  s = open("./serverX509Cert.pem").read()

  x509 = X509()

  x509.parse(s)

  certChain = X509CertChain([x509])

  s = open("./serverX509Key.pem").read()

  privateKey = parsePEMKey(s, private=True)

  verifierDB = VerifierDB("verifierDB")

  verifierDB.open()

  class Echo(LineReceiver):

      def connectionMade(self):

          self.transport.write("Welcome to the echo server!\r\n")

      def lineReceived(self, line):

          self.transport.write(line + "\r\n")

  class Echo1(Echo):

      def connectionMade(self):

          if not self.transport.tlsStarted:

              self.transport.setServerHandshakeOp(certChain=certChain,

                                                  privateKey=privateKey,

                                                  verifierDB=verifierDB)

          else:

              Echo.connectionMade(self)

      def connectionLost(self, reason):

          pass #Handle any TLS exceptions here

  class Echo2(Echo):

      def lineReceived(self, data):

          if data == "STARTTLS":

              self.transport.setServerHandshakeOp(certChain=certChain,

                                                  privateKey=privateKey,

                                                  verifierDB=verifierDB)

          else:

              Echo.lineReceived(self, data)

      def connectionLost(self, reason):

          pass #Handle any TLS exceptions here

  factory = Factory()

  factory.protocol = Echo1

  #factory.protocol = Echo2

  wrappingFactory = WrappingFactory(factory)

  wrappingFactory.protocol = TLSTwistedProtocolWrapper

  log.startLogging(sys.stdout)

  reactor.listenTCP(1079, wrappingFactory)

  reactor.run()

**安全考虑**

TLS Lite 是beta-quality code，尚未经过安全分析，风险自保