**Kerberos协议的原理**

2009/06/13 07:42   [[工作](http://blog.myspace.cn/fillon/category/20715.aspx) ]

这是MIT（Massachusetts Institute of Technology）为了帮助人们理解Kerberos的原理而写的一篇对话集。里面有两个虚构的人物：Athena和Euripides，通过Athena不断的构思和Euripides不断的寻找其中的漏洞，使大家明白了Kerberos协议的原理。  
Athena: 雅典娜，智慧与技艺的女神。  
Euripides:欧里庇得斯, 希腊的悲剧诗人。  
译文如下：  
第一幕  
在一个小工作间里。Athena和Euripides正在相邻的终端上工作。  
Athena: 嗨，这个分时\*作系统实在太慢了。我根本无法工作，因为每个人都登上去了。  
  
Euripides: 不要对我报怨。我只是在这工作。  
Athena: 你知道我们需要什么吗？我们需要给每一个人一台工作，这样大家就不会担心计算机的速度了。并且，我们需要一个网络把所有的计算机都联起来。  
Euripides: 好。那么我们差不多要一千台工作站？  
Athena: 差不多吧。  
Euripides: 你知道一台普通的工作站的硬盘有多大吗？那里放不下所有的软件。  
Athena: 我已经有主意了。我们可以把系统软件放到服务器上。当你登录到工作站的时候，工作站会通过网络与其中一台服务器上的系统软件联系。这样的设置让一组工作站都使用同一份系统软件，并且利于系统软件的升級。只需改动服务器就可以了。  
Euripides: 好的。个人的文件怎到办呢？在分时\*作系统上，我可以登录并从终端上取走我的文件。我能到工作站上取我的文件吗？我要象PC用户一样把我的文件放到磁盘上去吗？我希望不。  
Athena: 我想我们可以其它机器来存文件。你可以到任何一台机器上登录去取你的文件。  
Euripides: 打印怎么办呢？每个工作站都要有自已的打印机吗？谁来付钱？电子邮件呢？你怎么把邮件送到所有的工作站上去呢？  
Athena: 啊.....很明显我们没钱为每个人配一台打印机，但我们有专门的机器做打印服务。你把请求送到服务器，它就为你打印。邮件也可以这样做。专门有一台邮件服务器。你如果想要你的邮件，就联系邮件服务器，取走你的邮件。  
Euripides: 你的工作站系统听起来很不错。如果我有一台，你知道我要做什么吗？我要找出你的用户名，让我的工作站认为我就是你。然后我就去邮件服务器取走你的邮件。我会联上你的文件服务器，移走你的文件，然后－－  
Athena: 你能做得到吗？  
Euripides: 当然！这些网络服务器怎么会知道我不是你?  
Athena: 嗯，我不知道.我想我需要认真思考一下.  
Euripides: 好吧。你想出来后告诉我.

第二幕  
Euripides的办公室，第二天早上。Euripides坐在他的桌子旁边，读着他的邮件。Athena来敲门.  
Athena: 我已经想出怎样保护一个开放的网络系统，使象你那样不道德的人不能用别人的名字使用网络服务。  
Euripides: 真的吗？坐吧。  
她坐下了。  
Athena: 在我开始描述之前，我可以为我们的讨论先做一个约定吗？  
Euripides: 什么约定？  
Athena: 好，假设我这样说:"我想要我的邮件，于是我与邮件服务器联系，请求它把邮件送到我的工作站上来。"实际上我并没有联系服务器。我用一个程序来与服务器联系并取得我的邮件，这个程序就是这个服务的客户端。 但我不想每次与服务器交互的时侯说:"客户端怎样怎样".我只想说:"我怎样怎样,"记住，客户端在代表我做所有的事。这样可以吗？  
Euripides: 当然。没问题.  
Athena: 好。那么我要开始阐述我所解决的问题了。在一个开放的网络环境中，提供服务的机器必须能够识别请求服务的实体的身份。如果我去邮件服务器申请我的邮件，服务程序必须能够验证我就是我所申明的那个人。  
Euripides: 没错.  
Athena: 你可以用一个笨办法解决这个问题：服务器让你输入你的口令。通过输口令的办法我可以证明我是谁。  
Euripides: 那确实很笨拙。在像那样的系统里面，每一个服务器必须知道你的口令。如果网络有一千个用户,那每个服务器就要知道一千个口令。如果你想改变口令，你就必须联系所有服务器，通知它们修改口令。我想你的系统不会那么笨。  
Athena: 我的系统没那么笨。它是象这样工作的：不光人有口令，服务也有口令。每个用户知道他们自已的口令，每个服务也知道它自已的口令。有一个认证服务知道所有的口令，用户的和服务的。认证服务把口令保存在一个单独的中央数据库中。  
Euripides: 这个认证服务有一个名字吗？  
Athena: 我还没想好。你想一个吧？  
Euripides: 把死人送过冥河的人是谁？  
Athena: Charon?  
Euripides: 对，就是他。如果他不能证实你的身份的话，他就不会把你送过河。  
Athena: 你瞎编，是不是想重写希腊神话。Charon不关心你的身份，他只是确定你死了  
没有。  
Euripides: 你有更好的名字吗？  
停了一下。  
Athena: 没有，真的没有。  
Euripides: 好，那我们就把这个认证服务“Charon”。  
Athena: 好，我猜我该描述一下这个系统了吧，嗯?  
比如说我们想要一种服务：邮件。在我的系统里面你无法使用一种服务，除非Charon告诉服务你确实是你所申明的人。也就是说你必须得到Charon的认证才能使用服务。当你向Charon请求认证的时候，你必须告诉Charon你要使用哪一个服务。如果你想用邮件，你要告诉Charon。 Charon请你证明你的身份。于是你送给它你的密码。Charon把你的密码和它数据库中的密码相比较。如果相等，Charon就认为你通过了验证。 Charon现在就要让邮件服务知道你通过了验证。既然Charon知道所有服务的密码，它也知道邮件服务的密码。Charon把邮件服务的密码给你，你就可以使用这个密码使邮件服务相信你已通过验证。 问题是，Charon不能直接给你密码，因为你会知道它。下次你想要邮件服务的时候，你就会绕过Charon使用邮件服务而不需要认证。你也可以假装某人来使用邮件服务。 所以不是直接给你邮件服务的密码，Charon给你一张邮件服务的“票”。这张票含有你的名字，并且名字是用邮件服务的密码加密的。 拿到票，你就可以向邮件服务请求你的邮件。你向邮件服务提出请求，并用你的票来证明你的身份。 服务用它自已的密码来把票解密，如果票能被正确的解密，服务器将票里的用户名取出。服务把这个名字和随票一起送上的用户名进行比较。如果相符，服务器就认为你通过了验证，就把你的邮件发给你。  
  
你认为怎么样？  
Euripides: 我有些问题。  
Athena: 我猜到了。请讲。  
Euripides: 当服务解密一张票的时候，它如何知道它是被正确的解密的？  
Athena: 我不知道。  
Euripides: 也许你应该在票里包含有服务的名字。这样当服务解密票的时候，它就可以通过能否在票中找到自已的名字来判断解密是否正确。  
Athena: 很好。那票就应该是这个样子：  
(她把下面的东西写在了一张纸上)  
票-{用户名：服务名}  
Euripides: 那票就只包含用户名和服务名？  
Athena: 用服务的口令加密。  
Euripides: 我不认为这些信息就可以让票安全。  
Athena: 什么意思？  
Euripides: 假设你向Charon请求一张邮件服务的票。Charon准备了一张有你名字“tina”  
的票。假设在当票从Charon传给你的过程中我拷了一份。假设我让我的工作站相信我的用户名是”tina“。邮件客户程序认为我就是你。用你的名字邮件客户程序用偷来的票向邮件服务器提出请求。邮件服务器把票解密，认为它是合法的。票里的用户名和发送该票的用户名是匹配的。邮件服务器就会发给 我你的邮件。  
Athena: 喔!那可不太好。  
Euripides: 但是我想到了一个办法来解决这个问题。或者说部分解决。我想Charon应该在票中包含更多的信息。除了用户名，票还应包含请求票的用户的IP地址。这将给你增加一层安全性。 我来演示。假设现在我偷了你的票。这票有你工作站的IP地址，并且这地址配不上我的  
工作站的地址。用你的名字我把偷来的票送给邮件服务器。服务程序把用户名和网络地址从票中解出，并试图匹配用户名和网络地址。用户名匹配可网络地址不匹配。服务器拒绝了这张票，因为它明显是偷来的。  
Athena: 英雄，英雄!我怎么会没想到。  
Euripides: 好了，这就是我要表述的。  
Athena: 那么票应该是这个样子的。  
她把下面的东西写在了黑板上。  
票-{用户名：地址：服务名}  
Athena: 现在我真的很激动。让我们来建一个Charon系统看看它是否工作!  
Euripides: 没那么快。对于你的系统我还有些问题。  
Athena: 好吧。(Athena从她的椅子上探出了身子)快说。  
Euripides: 听起来好像每次我想要得到服务我都要去取一张新票。如果我整天的工作，我可能不只一次的要取我的邮件。我每次取邮件都要去取一张新票吗？如果真是这样，我不喜欢你的系统。  
Athena: 啊。。。我不明白为什么票不能被重用。如果我已经得到了一张邮件服务的票，我可以一次又一次使用它。当邮件客户程序用你的名字请求了服务，它就传了一份票的拷贝给服务。  
Euripides: 好一些。但我仍有问题。你似乎暗示我每次使用还没有票的服务时，我都必须给Charon我的密码我登录后想取我的文件。我向Charon请求我的票，这意味着我不得不使用我的密码。然后我想读我的邮件。又向Charon发一次请求，我又要输一次我的密码。现在假设我想把我的邮件送去打印。我又要向Charon发一次请求。你知道了吧？  
Athena: 啊，是的，我明白了。  
Euripides: 并且如果这还不够糟的话，想想看：它好像是这样，当每次你要向Charon认证的时候，你就要用明文在网络上传输你的口令。像你这样的聪明人可以监视网络并且得到别人的口令。如果我得到你的口令，我就可以用你的名字来使用任何服务。  
Athena叹了口气。  
Athena: 确实有严重的问题。我想我该回设计室去了。

|  |
| --- |
| 第三幕 第二天一早，Athena在咖啡间遇上了Euripides。在Euripides倒咖啡的时候，Athena拍了 拍Euripides. Athena: 我有了一个新的Charon的版本来解决我们的问题。 Euripides: 真的吗？好快呀。 Athena: 好，你看，这些问题困扰了我一夜。 Euripides: 一定是你良心发现了。我们去那边的小会议室吧？ Athena: 好的。 两人去了小会议室。 Athena: 我要重新描述问题，但我要根据我们的需要进行适当的转换。 Athena清了清嗓子。 Athena: 第一个限制：用户只输一次口令，在他们工作站启动的时候，这意味着当你需要申请新的服务的票时，不需输入你的口令。第二个限制：口令不能在网络上进行明文传输。 Euripides: 好的。 Athena: 我以第一项限制开始：你只需要输入你的口令一次。我创造了一个新的网络服务来解决这个问题。 它叫做“票据授权”服务，这个服务把Charon的票给用户。使用它必须要有票：票据授权的票。 票据授权服务其实只是Charon的一个版本，它可以存取Charon的数据库。它是Charon的一部分，可以让你通过票而不是口令来进行认证。 总之，认证系统现在是象这样工作的：你登录到一个工作站，用一个叫kinit的程序与Charon 服务器通讯。你向Charon证明你的身份，kinit程序取得一张票据授权票。 现在你想从邮件服务器上取你的邮件。你还没有邮件服务器的票，所以你用“票据授权”票去取邮件服务的票。你不需要使用口令去取新的服务票。 Euripides: 每次我想要另一种网络服务的时候，我都要去取一张“票据授权”票吗？ Athena: 不。记住，上次我们已经同意票是能被重用的。一旦你要用到票据授权票，直接用就可以了。 Euripides: 好，有道理。既然你能重用票，一旦你得到了某个服务的票，你就无需再去取了。 Athena: 对啊，那不好吗？ Euripides: 好的，我没话说，只要你在取得票据授权票的时候没有用明文在网上传输你的口令。 Athena: 如我所说，我已解决了这个问题。听起来好像是，当我说我要和Charon联系取 得票据授权票的时候，你就要在网络上传输明文密码。但其实不是这样的。 实际上是，当你用kinit程序取得票据授权票的时候，kinit没有把你的口令送 给Charon服务器，kinit只送你的用户名。 Euripides: 很好。 Athena: Charon用用户名去查找你的口令。然后Charon就会组一个包含票据授权票的包。在送给你之前，Charon用你的口令去把这个包加密。 你的工作站收到了包。你输入你的口令。kinit用你的口令对这个包进行解密。如果成功你就向Charon成功的进行了认证。你现在有了票据授权票，你可以用这张票来取得其它的票。 这些奇思妙想怎么样？ Euripides: 我不知道...我正在思考。你知道你的系统一部分工作得很好。你的系统只需要我认证一次。以后，Charon会给我服务的票而我需要关心。天衣无缝，天衣无缝。但服务票的设计还是有一些困扰我。服务票是可重用的。我同意它们应该能被重用，但重用的服务票，由于它们自身的性质，是非常危险的。 Athena: 什么意思？ Euripides: 这样看。假设你正在用一个不安全的工作站。在你登入后，你需要邮件服务票，打印票，和文件服务票。假设你无意中在你退出后留下了那些票。 现在假设我登录到那个工作站并且发现了那些票。我想制造一些麻烦，于是我就用你的名字登录了。既然那些票上是你的名字，那我就可以取你的邮件，打大量的文件。这些完全是因为这些票被偶然的放在了那里。 并且我还可以把这些票拷走，永远的使用它们。 Athena: 但是这很好解决。我们可以写一个程序，在用户退出的时候把票销毁掉，这些 票主不能再用了。 Euripides: 那么很明显你的统应该有一个票据销毁程序，让用户依赖这样的机制是非常愚蠢的。你不能指望用户在他们退出的时候会销毁票据。并且甚至不能依赖销毁票据本身，看下面的情况。 我有一个程序可以监视网络并且拷贝别人的服务票据。假设我想牺牲你。我等你登到工作站的时候，打开我的程序并拷贝一份你的票。 我等你退出并离开。我把我的工作站的地址调整为你登录时用的地址。我让工作站认为我是你。我有你的票，你的用户名，你的地址。我可以用这些票来使用你的服务。 你离开工作站时销毁你的票已没并系。这些我偷来的票可以一直使用下去，因为你现在的票并没有可以使用多少次的期限，或可以使用多长的时间。 Athena: 哦，我明白你所说的了！票不能是永远合法的，因为它可能是一个非常大的安全隐患。我们应该限制每一张票可以用多长的时间，也许可以给每张票设一个有效期。 Euripides: 非常正确。我想票需要增加两项信息：生存期表示票多长时间内是合法的，和一个时间标记来说明Charon是什么时候发出这张票的。 　Euripides走到了黑板写下了如下的内容： 　　票｛用户名：地址：服务名：有效期：时间戳｝ Euripides: 现在当服务解开票时，它检查票的用户名，地址是否与发送者匹配，然后它 用有效期和时间戳来检查票是否有效。  Athena: 很好。典型的票使用多长的有效期呢？ Euripides: 我不知道。也许是一个典型工作站的工作周期。就八小时吧。 Athena: 那如果我在工作站呆的时间超过八小时，所有的票将会失效。包括票据授权票。那我就要重新向Charon作认证，在八小时以后。 Euripides: 是不是不合理？ Athena: 我想不是。好我们就定下来吧－－票在八小时后失效。现在我有一个问题问你。假设我从网络上拷了你的票－－。 Euripides: (眨了眨眼睛）啊，Tina!你不会真的这样做吧？ Athena: 这只是为了讨论。我拷了你的票。现在我等你退出并离开。假设你有一个医生的约会或聚会要参加，你在两个小时后退出，并且你在退出之前销毁了你的票。 但我已经偷了你的票，它们还可以使用六小时。这给了我足够的时间用你的名义去取你的文件并打印一千份什么东西。 你看，时间戳工作的很好如果小偷选择在它失效以后来用的话。如果小偷能在它失效之前用...。 啊，好...当然，你是对的。 Athena: 我想我们遇上了一个大问题了。(她叹了口气) 停了一下。 Euripides: 我想这意味着你今晚要忙了。再来点咖啡？ Athena: 为什么不。  第四幕 第二天早上在Euripides的办公室。Athena来敲门。 Euripides: 你今早有黑眼圈了。 Athena: 好了，你知道的。又是一个漫漫长夜。 Euripides: 你解决了重演的问题了吗？ Athena: 我想是的。 Euripides: 请坐。 她坐下了。 Athena: 照旧，我重申一下问题。票是可重用的，在一个限定的时间内（八小时）。如果谁偷了你的票并在它失效之前使用，我们毫无办法。 Euripides: 确实如此。 Athena: 我们可以把这个问题理解为设计一种别人无法重用的票。 Euripides: 但这样的话你每次用新服务时都要取一张新票。 Athena: 对。但那是很笨的解决办法。（稍顿。）啊，我怎样继续我的讨论呢？（她沉思了一会儿）。 好的，我要重述一个问题，看有什么必须条件。网络服务必须能够证明使用票的人就是票上所申明的人。 我来顺着认证的过程再走一遍，这样我就可以演示我的解决方案。 我现在想用一个网络服务。我通过启动工作站上的客户端来使用它。客户端送三样东西给服务器：我的名字，我的工作站的网络地址，适当的服务票据。 这张票包含了申请这张票的人的名字和他（她）申请时所使用的工作站的地址。它也包含了票的有效期和时间戳。所有这些信息都被服务的密码加密了。 我们现在的认证模式基于以下的测试： 服务能对票解密吗？ 票在有效期以内吗？ 票中的名字和地址与申请者的名字和地址匹配吗？ 这些测试证明了什么？ 第一个测试证明了票是不是来自Charon.如果票不能被适当的解密，说明票不是来自真正的Charon. 真正的Charon会用服务的票来加密票。Charon和服务是唯一知道服务密码的两个实体。如果票被成功的解密，服务知道它来自于真的Charon.这个测试防止了有人伪造假票。第二项测试检查票是否在有效期以内。如果过期，服务拒绝。这项测试阻止使用旧票，因为票可能是偷来的。 第三项测试检查票的用户名和地址是否匹配请求者的用户名和地址。如果测试失败，说明使用者使用了别人的票。这张票当然被拒绝。如果名字和地址匹配，这个测试证明了什么？什么也没有。票可以被偷走，用户名和网络地址都可以被改变，如果需要的话。正如我昨天指出的那样，票可以在有效期内被盗用。 因为服务不能确定票的发送者是不是合法用户。 服务之所以无法判断是因为它没有与用户共享一个秘密。这样看。假如我正在埃尔斯诺 尔（哈姆雷特中的城堡）值勤，你打算来和我换班。但除非你说出正确的口令，否则我不 会与你换班的。我们共享了一个秘密。它可能是某人为所有值勤的人所设的。 于是昨晚我就在想，为什么Charon不能为合法用户与服务之间设一个口令呢？Charon发 一份口令给服务，同时发一份给用户。当服务从用户那里收到一张票，它可以用这个口令 检验用户的合法性。 Euripides: 等一下。Charon如何同时发两份口令？ Athena: 票据的拥用者从Charon的回应中得到口令，像这个样子： 她在黑板上写下了： Charon的回应－[口令｜票] 服务从票中获取口令。票的格式如下： 票－｛口令：用户名：地址：服务名：有效期：时间戳｝ 当你要请求服务时，客户端程序生成一个‘验证器’。验证器包含了你的名字和你工作站的地址。客户端用口令把这些信息加密，口令是你请求票据时得到的。 　验证器－｛用户名：地址｝用口令加密。 生成验证器以后，客户端把它和票一起送给服务。因为服务没有口令，所以它不能解密 验证器。口令在票中，于是服务先解开票。 解开票以后，服务得到以下的东西： 票的有效期和时间戳； 票的拥有者的名字； 票拥有者的网络地址。 口令。 服务检查票是否过期。如果一切正常，服务就用口令去解验证器。如果解密没有问题，服务将会得到一个用户名和网络地址。服务用它们去和票里的用户名和网络地址去匹配，如果正确，那么服务认为票的发送者确实是票的真实拥有者。 Athena暂停了一下，清了清喉咙，喝了点咖啡。 我认为口令验证器的机制解决了盗用的问题。 Euripides: 也许。但我想。。。攻击这个系统我必须有验证器。 Athena: 不。你必须同时拥有验证器和票。没有票，验证器是没有用的。解开验证器必须要有口令，服务必须解开票才会有口令。 Euripides: 好，我明白了，你是说当客户程序联系服务时，它同时送上票和验证器？ Athena: 是的，我就是这个意思。 Euripides: 如是真是这样，什么可以阻止我把票和验证器都偷走呢？我可以写一个程序，如果我拥有了票和验证器，我就可以一直使用它至有效期结束。我只需改变我的用户名和工作站的地址。不是吗？ Athena: (咬了咬她的嘴唇）是的。多沮丧啊。 Euripides: 等等，等等，等等！这不难解决。票在有效期内是可重用的，但那并不意味着验证器是可重用的。假设我们设计了验证器只可以被用一次。这可以吗？ Athena: 好，也许。我样来想一下，客户端程序生成验证器，然后把它和票一起送给服务。真的票和验证器比你拷贝的要先到。如果验证器只能被用一次，你的拷贝就失效了。 啊，这就对了。我样现在需要做的就是发明一和方法使得验证器只能被用一次。 Euripides: 没问题。我们把有效期和时间戳放在上面。假设每个验证有两分钟的有效期。当你想用一个服务时客户端生成验证器，标上当前的时间，把它和票一起送给服务。 服务器收到了票和验证器，服务器解开验证器，它检查验证器的时间戳和有效期。如果验证器还没失效，所有其它的检查都通过了，那么服务器就认为你通过了认证。 假设我通过网络拷贝了一份验证器和票，我必须改变我的工作站的网络地址和我的用户名，这差不多要用几分钟。那是非常苛刻的要求，我不认为是可能的，除非。。。 嗯，有一个潜在的问题。假设不是在网络的转输中拷贝到票和验证器，我拷贝了一份原始的从Charon而来的包，这个包是你向Charon请求时的回应。 这个包，有两个口令在里面：一个是你的，一个是服务的。服务的口令隐藏在票中，我取不到，但另一个呢？那个你用来生成验证器的？ 如果我得到了口令，我就用它来建自已的验证器，如果我能建自已的验证器，我就能攻破你的系统。 Athena: 这就是我昨晚所想的，但是当我顺着票的处理过程一想，发现那样偷走验证器是不可能的。 你在一台工作站坐下，用kinit程序得到你的票据授权票。kinit要求输入用户名，你输入以后，kinit把它送给Charon.Charon用你的名字查找你的口令，然后生成一张票据授权票。作为处理的一部分，Charon生成了一个你与票据授权服务共享的口令。Charon把口令和票据授权票一起送给你，并且在发关之前用你的口令将它加密。 Charon送出了包。某人取得了这个包，但他们无能为力因为它是用你的口令加过密的。特别是，无人可以偷走票据授权服务的口令。 kinit收到了票据包并要求你输入你的口令。如果你输入正确的口令，kinit解开包取出了口令。 现在你注意kinit的处理，你去取你的邮件。你打开邮件客户端。这个程序查找一张邮件服务的票但没有找到（你还没取过你的邮件）。客户端用票据授权票去申请一张邮件服务的票。 客户端为票据授权的过程生成了一个验证器，并用票据授权的口令把验证器加密。客户端把验证器送给了Charon，票据授权票，你的名字，你的工作站的地址，邮件服务的名字。票据授权服务收到了这些东西，并通过了认证检查。如果一切都通过，票据授权服务将会得到那个与你共享的口令。现在票据授权服务为你生成了一张邮件服务的票，在这个过程中生成了一个你与邮件服务共享的口令。票据授权服务把这些东西打成包送给你的工作站。包里有票和口令。在送包之前，票据授权服务用票据授权的口令把包加密。做完以后，包被送出去。 这样邮件服务票的包通过网络被送了出来。假设网络上的某人将它复制了一份。他不幸的发现包是用票据认证的口令加过密的。既然无法解密，他就不能得到邮件口令。没有口令，他就不能使用任何在网络上传送的邮件服务的票。 现在我觉得我们是安全的。你认为呢？   Euripides: 也许吧。 Athena: 也许！你就只会说这个吗！ Euripides: (大笑）别在意。你现在应该知道我处理问题的方式了。我猜我和你昨晚都工 作到了半夜。 Athena: 哼！ Euripides: 好的，大半夜。实际上，这个系统似乎是完全可行的。口令的方案解决了我 昨晚想到的一个问题：相互验证的问题。 稍顿。 我说一下好吗？ Athena: (有点冷淡）请便。 Euripides: 你真好。(Euripides清了清自已的嗓子）昨晚，当口令和验证器在我脑子里转的时候，我想去找出这个系统新的问题，我想我发现了一个很严重的问题。我下面就演示一下。 假设你厌倦了现在的工作，决定换一个。你想用公司的激光打印机打印求职信，把它们送给猎头和其它的雇主。于是你输入打印命令，命令去取得服务票，然后把票送到打印机。这是你认为它应该被送到的地方。实际上你并不知道你的请求被送到了正确的打印服务器。 假设一些无耻的人－－比如说你的老板－－调整了系统，把你的请求送到了他办公室的打印机。他的打印服务不关心票的内容。它告诉你的工作站服务已准备好打印你的文件。打印命令被送到了假的打印服务器，你有麻烦了。 我从相反的方向表达了相同的问题。用口令和验证器，Charon能够保护的它的服务器防止错误的用户使用，但它不能保护它的用户使用错误的服务器。系统需要为客户端程序提供一种验证服务器的方法，在它向服务器发送敏感信息之前。系统必须允许交互验证。 但口令的方案解决了这个问题。让我们回到打印服务器的场景。我想要打印客户程序确 认它送交的服务是合法的服务。 这就是程序要做的。我输入打印命令并给出一个文件名。这时我已经有了打印服务票和口令。客户程序用密码生成了一个验证器，然后把验证器和票送给了假设的打印服务器。客户端这时还没有送打印文件，它在等待从服务的返回。 真的服务收到票和验证器，把票解密并得到口令，然后用口令解开验证器。这样服务端做完了所有的认证。 测试已经确认了我的身份。现在服务程序要准备一个响应包来证实它自已的身份。它用口令加密了返回包，并把包送给了等待的客户端。 客户端收到了包并试图用口令把它解开。如果包被正确的解开得到了正确的服务器响应信息，客户端程序就知道了这个服务器是合法的服务器。然后这时客户端向它发出打印命令。 假设我的老板改变了一下系统使得他的打印机看起来好像是我想要用的那个。我的客户端送了票和验证器给它并等待它的响应。假冒的打印服务无法生成正确的响应因为它无法把票解开并得到口令。这样的话客户端就不会送打印命令给它因为客户端没有得到正确的 响应。最后客户端放弃等待并退出。我的打印没有完成，但至少我的求职信不会放在我的 对头的桌子上。 好啊，我想我们有了Charon认证系统的坚实的基础。 Athena: 也许。不管怎么说，我不喜欢Charon这个名字。 Euripides: 你不喜欢吗？什么时候？ Athena: 我从来都不喜欢，因为它的名字听起来没意义。有一天我和我荷迪斯（冥王)叔叔 谈到了这个，他推荐了另一个名字：冥王的三个头的看门狗。 Euripides: 啊，你是说“Cerberus". Athena: 你说什么语言啊！"Cerberus"实际上是。。。 Euripides: 哦，不叫这个吗？ Athena: 当然，谁让你是罗马人！而我是希腊人，它是一条希腊的看门狗，它的名字是” Kerberos“，”Kerberos“是‘K’打头的。 Euripides: 好吧，好吧，别发火。我同意这个名字。实际上，它有一个好的脖环。再见 吧，Charon,欢迎你，Kerberos. 后记 这篇对话是于1988年写的，是为了帮助读者理解Kerberos V4的运行方式。经过了这么多年，它仍然非常好的服务于此。当我把这篇文章转换成HTML的时候，我惊讶的发现这个文档对Kerberos　V5仍然非常有用。虽然很多东西改变了，但核心概念并没有变。实际上，Kerberos V5对Kerberos只做了两处改变。 第一处改变是因为意识到验证器用少于五分钟的有效期不足以防止攻击者进行重演，如果攻击者是用一个程序自动的截取票和验证器并进行重演的话。 在Kerberos V5中，验证器真正的只能用一次因为服务器用‘重演缓冲区’保存了最近一次提交的验证器的信息。如果攻击者试图截取验证器并重用它，‘重演缓冲区’会发现验证器已经被提交了。 第二个主要改变是Kerberos送给kinit服务票的时候，票不再是用用户的口令加密。它已经用票据授权服务的口令加过密了。票据授权服务的票被用来获取其它票的时候，它直接就被传输了。因此票不需要再用用户的口令加密一次。（服务器响应的其它部分，如口令，仍然是用用户的口令加密的。） 一个类似的改变也应用到票据授权服务协议；从票据授权服务返回的票也不再用票据授 权服务的口令来加密了，因为它所包含的票已经被对应的服务的口令加过密了。举例来说， Kerberos V4的包像这样： KDC\_REPLY = {TICKET, client, server, K\_session}K\_user 意思是：｛｝中的内容是用K\_user来加密的。 TICKET = {client, server, start\_time, lifetime, K\_session}K\_server 在Kerberos V5中，KDC\_REPLY现在看起来像这样： KDC\_REPLY = TICKET, {client, server, K\_session}K\_user (注意：票已经不再用K\_user来加密了) 当然，Kerberos V5中还有许多新特性。用户可以在另一个网络中安全的提交他们的票；并且，用户可以把他们的一部分认证权转给服务器，这样服务器就可以作为用户的代理。其它的新特性包括：用更好的加密算法替换了DES加密算法，如三重DES加密。读者如果对V4与V5的变化感兴趣的话，可以读一下"The Evolution of the Kerberos Authentication System",作者是Cliff Neumann和Theodore Tso. 我希望你能对这篇介绍Kerberos协议的文章感兴趣。我祝愿你在未来的探索中更进一步。 |