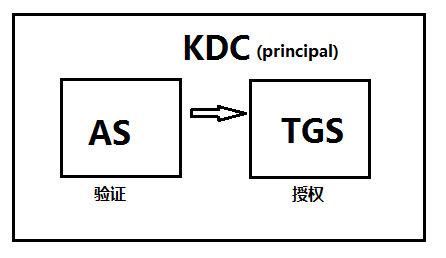
# Kerberos协议问题

## 快速介绍

Kerberos 是Windows活动目录中使用的客户/服务器认证协议，为通信双方提供双向身份认证。相互认证或请求服务的实体被称为委托人（principal）。参与的中央服务器被称为密钥分发中心（简称KDC）。KDC有两个服务组成：身份验证服务（Authentication Server，简称AS）和票据授予服务（Ticket Granting Server，简称TGS）。在Windows域环境下，身份验证服务和票据授予服务可同时运行在任何可写域控服务器上。（PS:Kerberos其实是希腊神话中看守地狱大门的三头猛犬的名字。）

总结一下，就是在域控上有一个协议运行机制，简称为KDC，KDC只负责两种服务，一个是验证服务（AS），一个是票据授权（TGS）



## Kerberos的作用

Kerberos是由麻省理工学院研发的，是一种基于第三方可信主机的计算机网络协议，它允许两个实体之间在非安全网络环境（可能被窃听、被重放攻击）下以一种安全的方式证明自己的身份。

### 这个协议运用于的情况是A对B发送了一个访问请求，那么A怎么才能向B证明自己就是A呢？

方式一：A与B共享一个key，当A向B提出请求时，发送key，B验证key，正确了便能证实A的身份

问题：网络环境默认不安全，那么key传输过程中就有可能被中间人攻击或者窃听， 所以不安全

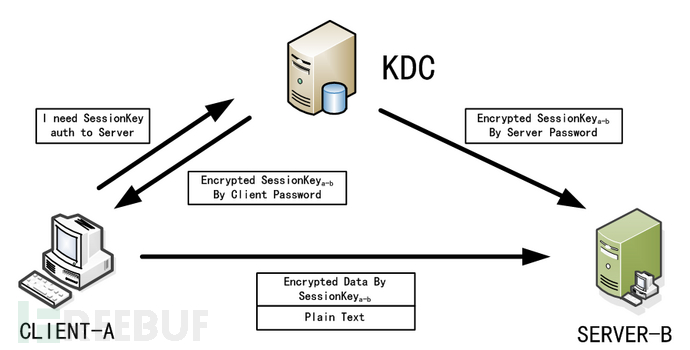
方式二：A与B共有一个key，A向B发送一段用key加密一段明文生成的密文，同时发送这段明文，B接收到后用key解密密文与接收的明文做对比，相同的话则能验证A的身份

问题：但如果该认证请求被窃听，攻击者能得到加密后密文和加密前的明文，只要时间允许，总能推导出密钥的数值，也就是秘密肯定会被窃取。所以密码界有个规定，长期存在的密钥不适合在网络中传输，否则总会有被窃取的风险。

这样问题就来了，不能使用长期密钥，那么谁来提供短期密钥呢？这就诞生了第三方可信机构KeyDistribution Center，KDC

也就是kerberos协议的运作目的，向AB双方提供短期密钥，用于双方的请求认证

A先去向KDC申请本次向B认证需要的Key，由KDC将Key分发给A和 B，最后A就可以利用这个Key向B进行认证了。为了Key不被窃取，该 Key用A和B自身的密码分别加密



这里会有两个问题：

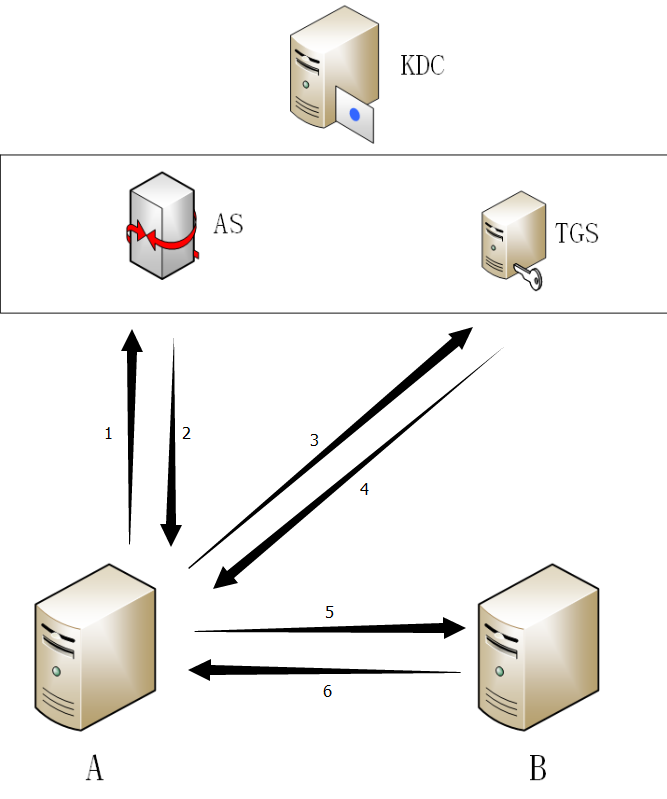
（1）A向KDC申请Key，它可能很快就收到Key，但B可能因为网络环境原因，很晚或者根本收不到Key，这样就导致认证无法进行，解决办法就是，KDC将两份加密的Key都发给A，由A在向B发出认证请求时将原本属于B的Key一同发送给B

（2）A提出Key的申请时，KDC凭什么就生成了Key给了A，也就是说，KDC缺乏对A的认证，所以在分发Key之前，KDC需要增加对A的认证，解决办法就是，将KDC机构分成两部分



了对短期有效的Key进行验证，Kerberos协议要求系统内的所有主机基于时间同步，所以 A向B进行认证就不用Key加密一段密文后还得发送一段明文过去，直接用Key加密当前时间即可。

那么，我们就可以叙述出kerberos的认证框架了



第①步：KRB\_AS\_REQ：A发送Authenticator向KDC的AS服务认证自己的身份（通过提供自身密码加密的一个时间戳TimeStamp）

第②步：KRB\_AS\_REP：AS通过KDC数据库中存储的A密码的副本，解密收到的Authenticator1，如果解密出的TimeStamp符合要求，则AS服务认为A就是所谓的A。认证成功后，AS服务生成一个短期有效的Keya-kdc，将该Key使用A的密码副本加密成密文1，另外将Keya-kdc连同时间戳标志（控制该Key的有效时间）通过TGS服务的密码也就是KDC的密码加密为密文2（称为TGT），将这两个密文组合成KRB\_AS\_REP返回给A

第③步：KRB\_TGS\_REQ：A在接收到KRB\_AS\_REP后，首先使用自身密码解密密文1得到Keya-kdc，此时需要注意的是，密文2（TGT）是被KDC的密码加密的，所以A无法解密（这也是Kerberos协议设计的精妙之处，既解决了Server端（TGS相对于A也称之为Server端）无法及时接收Key的问题，又不怕Client-A对该TGT的伪造，因为A不知道Server端的密码）

得到Keya-kdc后，A利用Keya-kdc加密时间戳生成Authenticator2用于向TGS申请A与B进行认证所需的Keya-b，连同刚才KRB\_AS\_REP接收的TGT一同组合成KRB\_TGS\_REQ发送给TGS

第④步：KRB\_TGS\_REP：TGS在接收到KRB\_TGS\_REP之后，利用KDC密码解密TGT获得本来就该发送给自己的Keya-kdc，然后用其解密KRB\_TGS\_REQ中的Authenticator2得到A发送过来的时间戳，如果时间戳符合要求，则生成一个短期有效的Keya-b，注意此时利用Keya-kdc将Keya-b加密为密文1，然后利用B的密码将Keya-b加密为密文2（称为ServiceTicket），两个密文一同构成KRB\_TGS\_REP返回给A

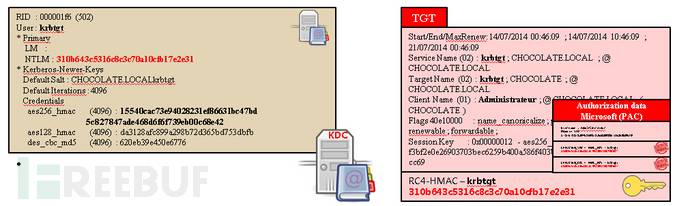
第⑤步：KRB\_AP\_REQ：A在接收到KRB\_TGS\_REP之后，首先使用缓存的Keya-kdc将密文1中的Keya-b解密出来，然后利用Keya-b加密时间戳生成Authenticator3用于向B进行对自身的验证，另外，和刚才TGT一样，密文2也就是ServiceTicket是用B的密码加密的，所以A无法解密，也就无法伪造，这也同样解决了在三方认证中作为Server端的B无法及时接收Key的问题，又不怕A对ServiceTicket的伪造

第⑥步：KRB\_AP\_REP：B受到KRB\_AP\_REQ之后，利用自身密码解密ServiceTicket，得到Keya-b，然后用Keya-b解密Authenticator3得到时间戳，验证A的身份

至此，kerberos协议验证结束

**Kerberos信任完全依赖于KDC密码(来源Freebuf)**

Kerberos协议是无状态的，因此密钥分发中心和票据授予服务并没记录以前的交互信息。因此票据授予服务所需使用的全部信息都位于TGT票据中。因为TGT使用KRBTGT的密码加密过，理论上讲网络上只有两方能够解密TGT：颁发票据的KDC和接受票据并创建访问网络资源的服务票据的票据授予服务。这种情况让KRBTGT成为系统中最重要的密码。最终结果是只要TGT被krbtgt账户密码正确地加密，TGT中的所有信息都是可信的。



**小科普**

krbtgt账户：每个域控制器都有一个“krbtgt”的用户账户，是KDC的服务账户，用来创建票据授予服务（TGS）加密的密钥。

## 一个看似十分严谨的认证机制，然而却包含着很多隐患

### Kerberos中包含的问题及利用

问题一：**pass-the-hash** and **pass-the-ticket**

<http://drops.wooyun.org/tips/11631>

<http://www.tuicool.com/articles/UFraUnQ>

Kerberos协议从开发到使用其实是两个版本，一个是原始的MIT版本，另一个便是投入使用的微软实现的kerberos版本，也就是现在常说的kerberos协议

这儿就有了一个问题，因为微软对kerberos的函数做出了修改，导致底层安全性降低，最直观的就是在kerberos认证过程中的长期密钥并没有经过加盐运算，而是直接明文经过Unicode转码，然后MD4加密，生成了NT-HASH，而这就有了一个问题，原始版本中加盐是为了在涉及密钥的操作时，必须要经过实际的口令验证，而当密钥不加盐时，我们就可以直接使用密码的散列，达到验证的目的，也就是所谓的**pass-the-hash**

**再次看上面的那个Freebuf小科普**

因为协议无状态导致不记录交互信息，而是完全信任证书，这就导致了一个问题，我们可以从已经经过123步验证，获得TGT证书的计算机上，把TGT证书导出，然后导入到其余的计算机上，来维持票据中记录的认证权限，也就是**pass-the-ticket**

问题二：密码不自动更新

KDC具有一个控制账户，krbtgt，而这个账户是微软活动目录中，唯一一个不会自动更新的密码，就因为这样，krbtgt的密码有可能始终如一（除非对整域环境进行大的变动）

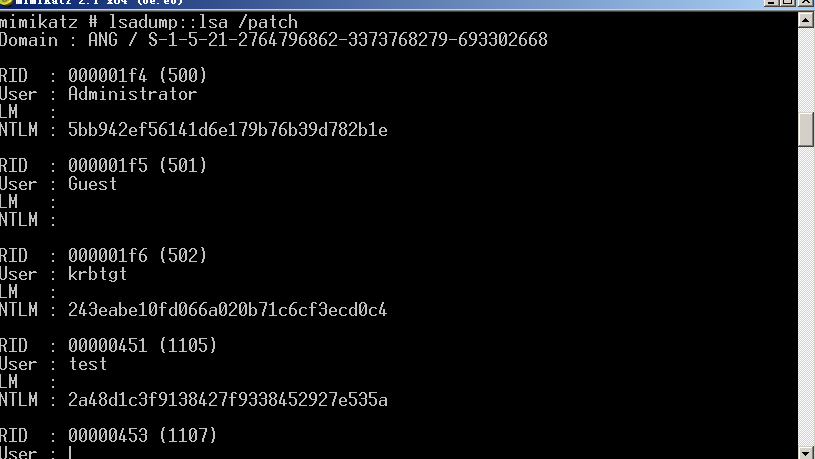
而密码不变会带来什么问题呢？很简单，krbtgt是KDC的控制账户，当你拿到了krbtgt的密码散列，就能控制住KDC，而控制了KDC，便意味着可以为任意计算机授予票据，那这样我们就能这么想，我们拿了一个域控，记录下krbtgt的密码散列，然后利用域控本身的权限去对KDC生成授予票据的请求（用于访问TGS），运用krbtgt的密码散列去通过这个请求，就这样，我们就拥有了一份权限是域控的TGT文件，这样的话，当域控的密码改了，我们依然能重新获得域控权限

## 实验验证

而这一个重要的就是我们需要krbtgt用户的ntlm哈希

然后就是域用户的SID和域管理员名称

登陆域控，在域控中抓取



记录下ntlm的hash，

mimikatz # kerberos::purge

mimikatz # kerberos::golden /admin:Administrator /domain:pentstlab.com /sid:S-1-5-21-3883552807-251258116-2724407435 /krbtgt:6a8e501fabcf264c70ef3316c6aab7dc /ticket:Administrator.kiribi

mimikatz # kerberos::ptt Administrator.kiribi

mimikatz # kerberos::tgt

到现在，我们又重新拥有域管理员权限了，可以验证下

net use \\WIN-0DKN2AS0T2G