Kerberoasting Without Mimikatz

Published: 2016-11-01 · Archived: 2026-04-05 19:14:37 UTC

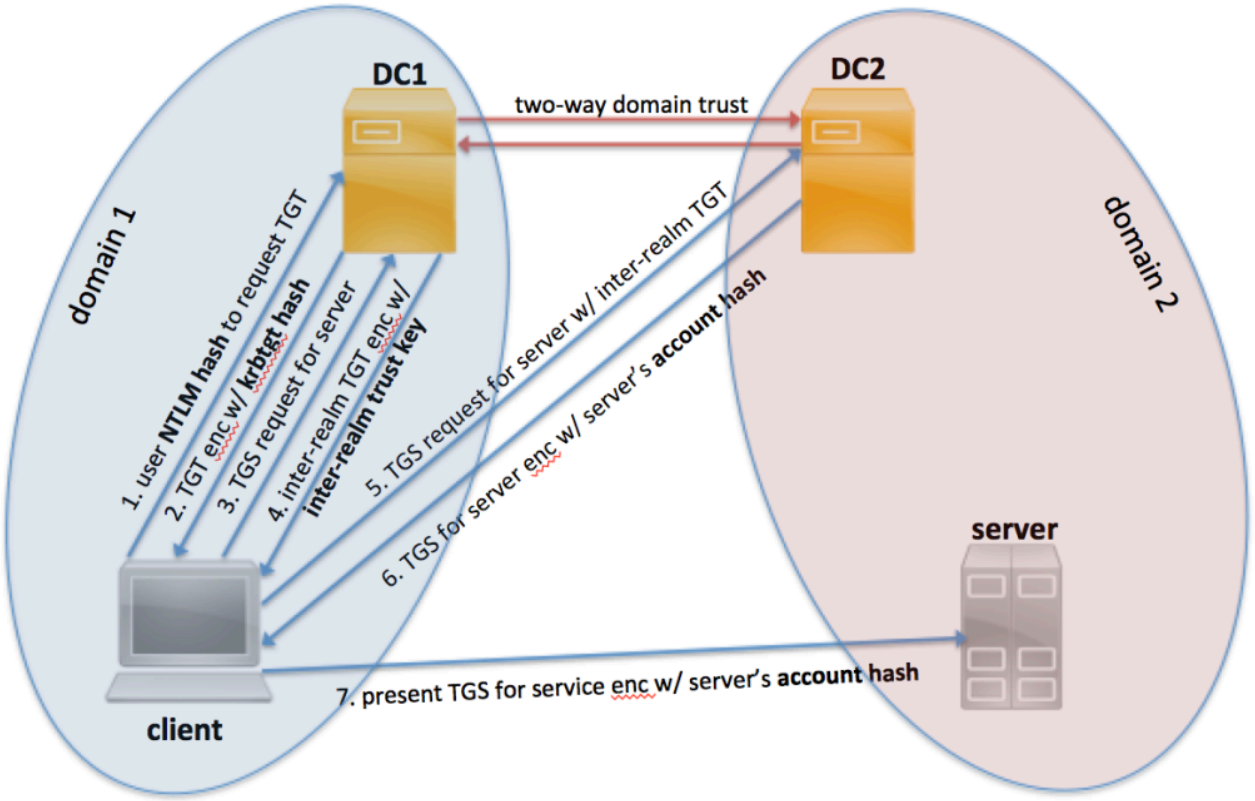
Just about two years ago, Tim Medin presented a new attack technique he christened "Kerberoasting". While we didn't realize the full implications of this at the time of release, this attack technique has been a bit of a game changer for us on engagements. More and more attention has been brought to Kerberoasting recently, with @mubix releasing a three part series on the topic, Sean Metcalf covering it several times, and @leonjza doing a detailed writeup as well.

Thanks to an awesome PowerView pull request by @machosec, Kerberoasting is easier than ever using pure PowerShell. I wanted to briefly cover this technique and its background, how we've been using it recently, and a few awesome new developments.

Kerberoasting Background

I first heard about Kerberoasting from Tim at SANS HackFest 2014 during his "Attacking Kerberos: Kicking the Guard Dog of Hades" talk (he also released a Kerberoasting toolkit here). I'll *briefly* paraphrase some technical detail of the attack, but I highly recommend you read Tim's slides and/or Sean's explanation for more detail. There's also an excellent page of Microsoft documentation titled "Kerberos Technical Supplement for Windows" which finally clarified a few points involved in this process that were fuzzy to me.

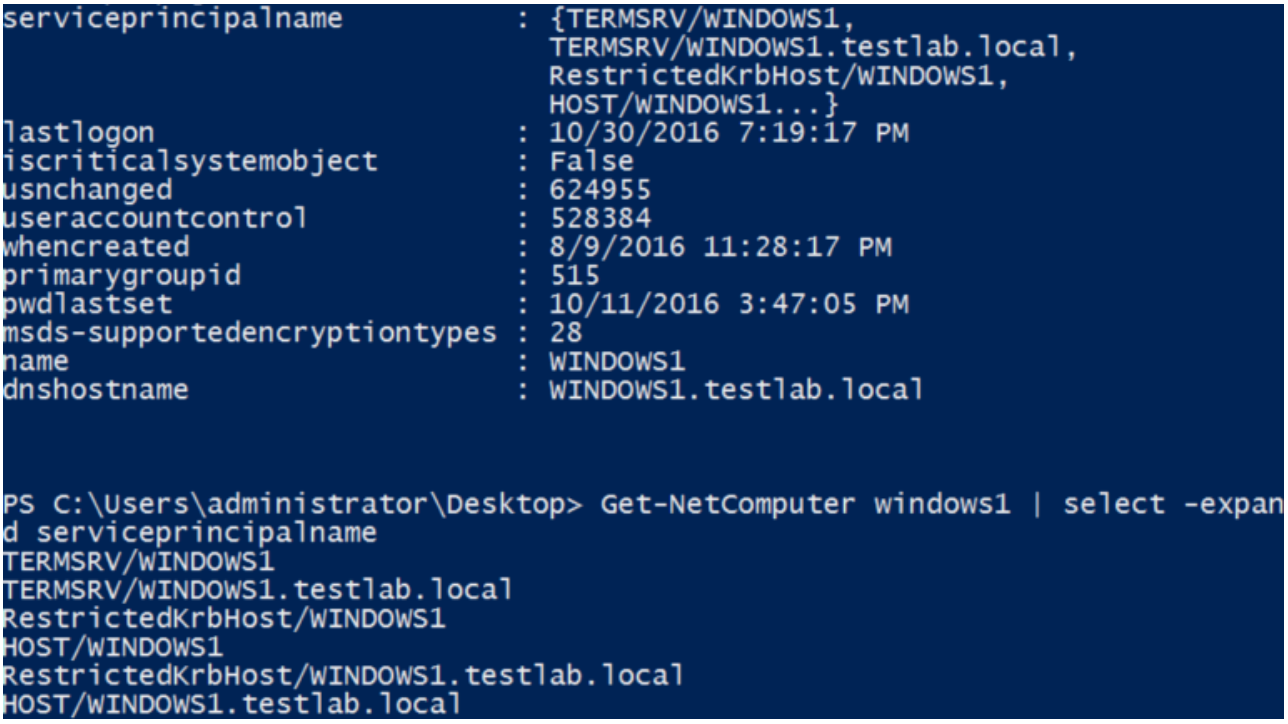
Here's my version of the obligatory "this is how kerberos works" graphic:



As far as how Kerberoasting fits into this process, this is how I understand it (if I am mistaken on some point please let me know!): after a user authenticates to the key distribution center (KDC, which in the case of a Windows domain is the domain controller) they receive a ticket-granting-ticket (TGT) signed with the domain krbtgt account that proves they are who they say they are. The TGT is then used to request service tickets (TGS) for specific resources/services on the domain. Part of the service ticket is encrypted with the NTLM hash of the target service instance. So how does the KDC determine exactly what key to use when encrypting these service tickets?

The Windows implementation of the Kerberos protocol uses service principal names (SPNs) to determine which service account hash to use to encrypt the service ticket. There are two "types" of service principal names in Active Directory: "host-based" SPNs that are linked to a domain **computer** account and "arbitrary" SPNs that are usually (but not always) linked to a domain **user** account.

As Microsoft explains, "*When a new computer account is created in Active Directory, host-based SPNs are automatically generated for built-in services…In reality, SPNs are only created for the HOST service and all built-in services use the HOST SPN*". Put another way, "*The HOST service represents the host computer. The HOST SPN is used to access the host computer account whose long term key is used by the Kerberos protocol when it creates a service ticket*". Here's an example of a default **computer** account in my test domain:



You can see the HOST/WINDOWS1 and HOST/WINDOWS1.testlab.local SPNs for the WINDOWS1$ computer account. When a domain user requests access to \\WINDOWS1.testlab.local\C$, the KDC maps this request to the HOST/WINDOWS1.testlab.local SPN, indicating that the WINDOWS1$ machine account NTLM hash (which is stored both on WINDOWS1 locally and the NTDS.dit Active Directory database on the DC/KDC) should be used to encrypt the server part of the service ticket. The signed/encrypted ticket is then presented to WINDOWS1.testlab.local, which is responsible for determining whether the requesting user should be granted access.

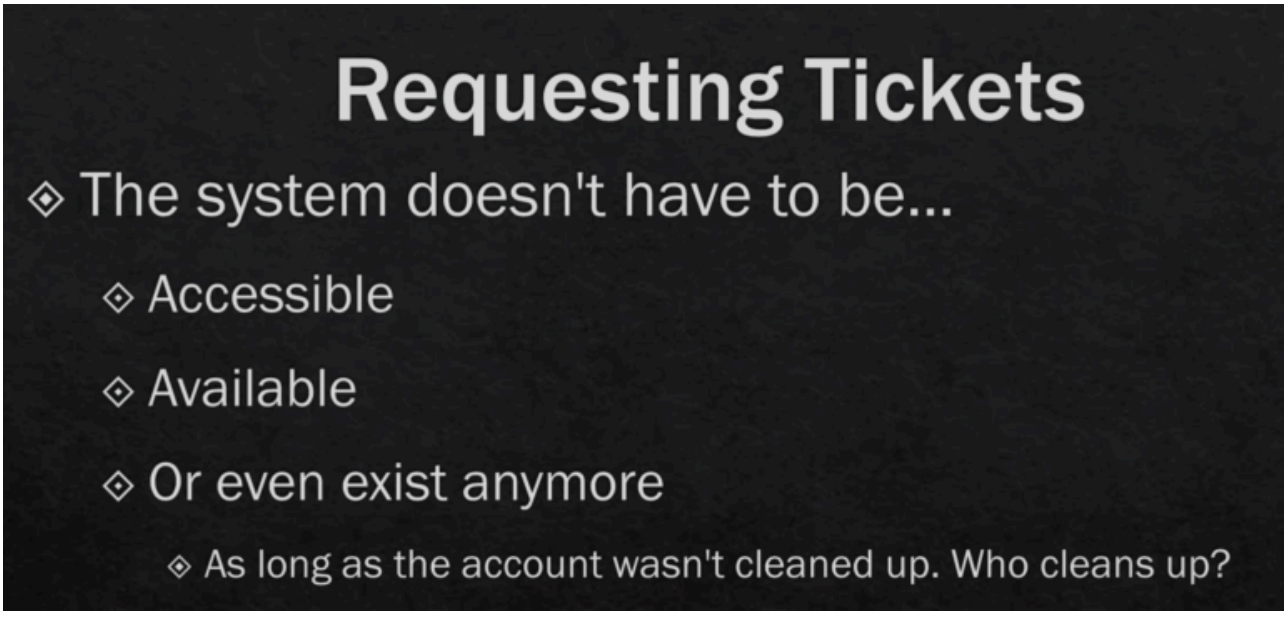
From the Kerberoasting perspective, we generally don't care about host-based SPNs, as a computer's machine account password is randomized by default and rotates every 30 days. However, remember that arbitrary SPNs can also be registered for domain **user** accounts as well. One common example is a service account that manages several MSSQL instances; this user account would have a <MSSQLSvc/HOST:PORT> SPN for each MSSQL instance it's registered for stored in the user's **serviceprincipalname** attribute (Sean keeps an updated list of SPNs here). If we have an arbitrary SPN that is registered for a domain user account, then the NTLM hash of that user's account's *plaintext* password is used for the service ticket creation. **This is the key to Kerberoasting.**

Obligatory "So Why Does This Matter?"

Because of how Kerberos works, any user can request a TGS for any service that has a registered SPN (HOST or arbitrary) in a user or computer account in Active Directory. Remember that just requesting this ticket doesn't grant access to the requesting user, as it's up to the server/service to ultimately determine whether the user should be given access. Tim realized that because of this, and because part of a TGS requested for an SPN instance is encrypted with the NTLM hash of a service account's plaintext password, **any user can request these TGS tickets and then crack the service account's plaintext password offline**, without the risk of account lockout!

To reiterate, any domain user account that has a service principal name set can have a TGS for that SPN requested by **any user in the domain**, allowing for the offline cracking of the service account plaintext password! This is obviously dependent on a crackable service account plaintext, but luckily for us service accounts tend to often have simple passwords that change very infrequently. ¯_(ツ)_/¯

As an added bonus, Tim mentions on slide 18 of his presentation deck:



¯_(ツ)_/¯

"Old School" Kerberoasting

Tim's outlined approach/toolkit used a combination of toolsets to request tickets, extract them from memory (using Mimikatz), and transform them into a crackable format. In general, the process (up until recently) went as follows:

- Enumerate the domain accounts with SPNs set- either with Tim's GetUserSPNS.ps1 script, Sean's Find-PSServiceAccounts.ps1 script, or PowerView's "Get-NetUser -SPN".
- Request TGSs for these specific SPNs with the builtin Windows tool setspn.exe or the .NET System.IdentityModel.Tokens.KerberosRequestorSecurityToken class in PowerShell.
- Extract these tickets from memory by invoking the **kerberos::list /export** Mimikatz command , with the optional base64 export format set first. The tickets were then downloaded, or the base64-encoded versions pulled down to the attacker's machine and decoded.
- Begin offline password cracking with Tim's tgsrepcrack.py, or extract a crackable hash format from the raw ticket with John the Ripper's kirbi2john.py.

xan7r branched Tim's toolset and added an autokerberoast.ps1 script that automated large components of this process. Also, @tifkin_ wrote a Go version of a TGS cracker that functioned a bit faster than the original Python version.

"New School" Kerberoasting

A few recent(ish) things really simplified our usage of Kerberoasting on engagements. First, Michael Kramer added the KRB5TGS format to John the Ripper in September of 2015. Second, @Fist0urs committed the same algorithm to Hashcat in Febuary 2016, opening the door for GPU-based cracking of these tickets. This was really a watershed for us, as it greatly expanded the range of service account passwords we could crack. And finally, Matan Hart (@machosec)'s pull request to PowerView removed the Mimikatz requirement.

@machosec realized that .NET class KerberosRequestorSecurityToken used in previous approaches also had a GetRequest() method, which returns the raw byte stream of the Kerberos service ticket. With a bit string manipulation, Matan was able to easily extract out the encrypted (i.e. the crackable hash component) of the TGS. We are now no longer dependent on Mimikatz for ticket extraction!

I recently rolled the necessary functions into a single, self-contained script that contains the necessary components from PowerView (this has also been updated in Empire). We are currently in the process of refactoring large components of PowerSploit, and the updated functions will be posted here after the changes are published. This custom-rolled script includes the **Invoke-Kerberoast** function, which wraps the logic from **Get-NetUser -SPN** (to enumerate user accounts with a non-null servicePrincipalName) and **Get-SPNTicket** to request associated TGS tickets and output John and Hashcat crackable strings. For now, here's what the output of the script looks like:

```
PS C:\Users\administrator\Desktop> Invoke-Kerberoast | fl


SamAccountName       : SQLService
DistinguishedName    : CN=SQLService,CN=Users,DC=testlab,DC=local
ServicePrincipalName : MSSQLSvc/PRIMARY.testlab.local:1433
Hash                 : $krb5tgs$unknown:30FFC786BECD0E88992CBBB017155C53$0
                       343A9C8A7EB90F059CD92B5271414AB510EFE9379DE1BACD220
                       67FE4909DE3D2D860320586DED3EF4DBE25A0D73329E4E47D3F
                       D043D1BD5CCA668243186322934 76A5E741A444F0FA874C8DBF
                       8059850F14929F52DAACFD13BEEA754B27A0B190AC7AB53FC33
                       8F581E8AB76D002DF1E4619920AA4B372219DAE3256BF8D38CB
                       978ACE111ADE5ACB2F1ED9DDC85CC3E8A507E90F57ECE329A9A
                       E18F51C918DF9334BEC79C01C4DD4341BD2E1C1666BB6AAB2F0
                       39046CC4A24B71A6640A4E0C7D8C012F8864079D0844D5869F7
```

It also works across domains!

```
PS C:\Users\administrator\Desktop> Invoke-Kerberoast -Domain dev.testlab.lo
cal | fl


SamAccountName       : SQLSVC
DistinguishedName    : CN=SQLSVC,CN=Users,DC=dev,DC=testlab,DC=local
ServicePrincipalName : MSSQLSvc/secondary.dev.testlab.local:1433
Hash                 : $krb5tgs$unknown:ECF4BDD1037D1D9E2E091ABBDC92F00E$0
                       F3A4AB1F7523B1D182ECD0C934F97BF96EB03A55439064CFC33
                       1F9ACC6FADA95927304ECE78B8B358EBC906629590BBAE527DC
                       2B87A28D0A72E664CE79FB9A3B8C7EE92FF11A7FED3745C7D52
                       0E38703F2CF63D9AAA299F831AFA2E2194AFD72EDC58568318A
                       835A0FB98353FCE6F5D599C4F7F9DAA8D6EA1B65414AF87A9B7
                       F1371A45BBF6D6C868E9DDC804DD9F3136F7B3BB75912569CDF
                       43C613458F8767DB342CD9E6E16F6F29910D83E892B3290F5C2
                       1F4EFC68CFE3322C31513A8C51C9001A20A75156EC61A79520F
```

By default, the John format is output, but **-OutputFormat Hashcat** will output everything Hashcat-ready. Note that the **-AdminCount** flag only Kerberoasts accounts with AdminCount=1, meaning user accounts that are (or were) 'protected' and, therefore, almost always highly privileged:

```
PS C:\Users\administrator\Desktop> Invoke-Kerberoast -AdminCount -OutputFor
mat Hashcat | fl


SamAccountName       : SQLService
DistinguishedName    : CN=SQLService,CN=Users,DC=testlab,DC=local
ServicePrincipalName : MSSQLSvc/PRIMARY.testlab.local:1433
Hash                 : $krb5tgs$23$*ID#124_DISTINGUISHED NAME: CN=fakesvc,
                       OU=Service,OU=Accounts,OU=EnterpriseObjects,DC=prod
                       dfs,DC=pf,DC=fakedomain,DC=com SPN: E3514235-4B06-1
                       1D1-AB04-00C04FC2DCD2-ADAM/NAKCRA04.proddfs.pf.fake
                       domain.com:50000 *30FFC786BECD0E88992CBBB017155C53$
                       0343A9C8A7EB90F059CD92B5271414AB510EFE9379DE1BACD22
                       067FE4909DE3D2D860320586DED3EF4DBE25A0D73329E4E47D3
                       FD043D1BD5CCA668243186322934 76A5E741A444F0FA874C8DB
                       F8059850F14929F52DAACFD13BEEA754B27A0B190AC7AB53FC3
```

And here's how the [updated Empire module](#) looks:

```
(Empire: powershell/credentials/invoke_kerberoast) > set Agent EZVY2HXN
(Empire: powershell/credentials/invoke_kerberoast) > execute
(Empire: powershell/credentials/invoke_kerberoast) >
Job started: W4XBE2



SamAccountName       : SQLService
DistinguishedName    : CN=SQLService,CN=Users,DC=testlab,DC=local
ServicePrincipalName : MSSQLSvc/PRIMARY.testlab.local:1433
Hash                 : $krb5tgs$unknown:30FFC786BECD0E88992CBBB017155C53$0343A9C
                       8A7EB90F059CD92B5271414AB510EFE9379DE1BACD22067FE4909DE3D
                       2D860320586DED3EF4DBE25A0D73329E4E47D3FD043D1BD5CCA668243
                       18632293476A5E741A444F0FA874C8DBF8059850F14929F52DAACFD13
                       BEEA754B27A0B190AC7AB53FC338F581E8AB76D002DF1E4619920AA4B
                       372219DAE3256BF8D38CB978ACE111ADE5ACB2F1ED9DDC85CC3E8A507
                       E90F57ECE329A9AE18F51C918DF9334BEC79C01C4DD4341BD2E1C1666
                       BB6AAB2F039046CC4A24B71A6640A4E0C7D8C012F8864079D0844D586
                       9F79B9F921281E501A2D7CAED59B44579AEAF2E4CCE638A0CF5252396
                       8FFD6C044D0101B40BE6318287A0A95A53EF031B80F63297F5568524B
                       695FCF15D576A1F7F74D12D3D20A88AC00DF32F0AAD6BADF96B0C28A4
```

Note that for non-Empire weaponizations, as PSObjects are output, you will need to pipe the results to **Format-List** or **ConvertTo-Csv -NoTypeInformation** in order to preserve the information you want displayed. You can then crack these tickets as [@mubix](#) described in [his third post](#).

Again, the self-contained, PowerShell 2.0-compliant script is on my [Gists here](#). Hopefully this is as much use to you as it has been for us over the past few months!

Source: https://blog.harmj0y.net/powershell/kerberoasting-without-mimikatz/