When Kirbi walks the Bifrost

By Cody Thomas

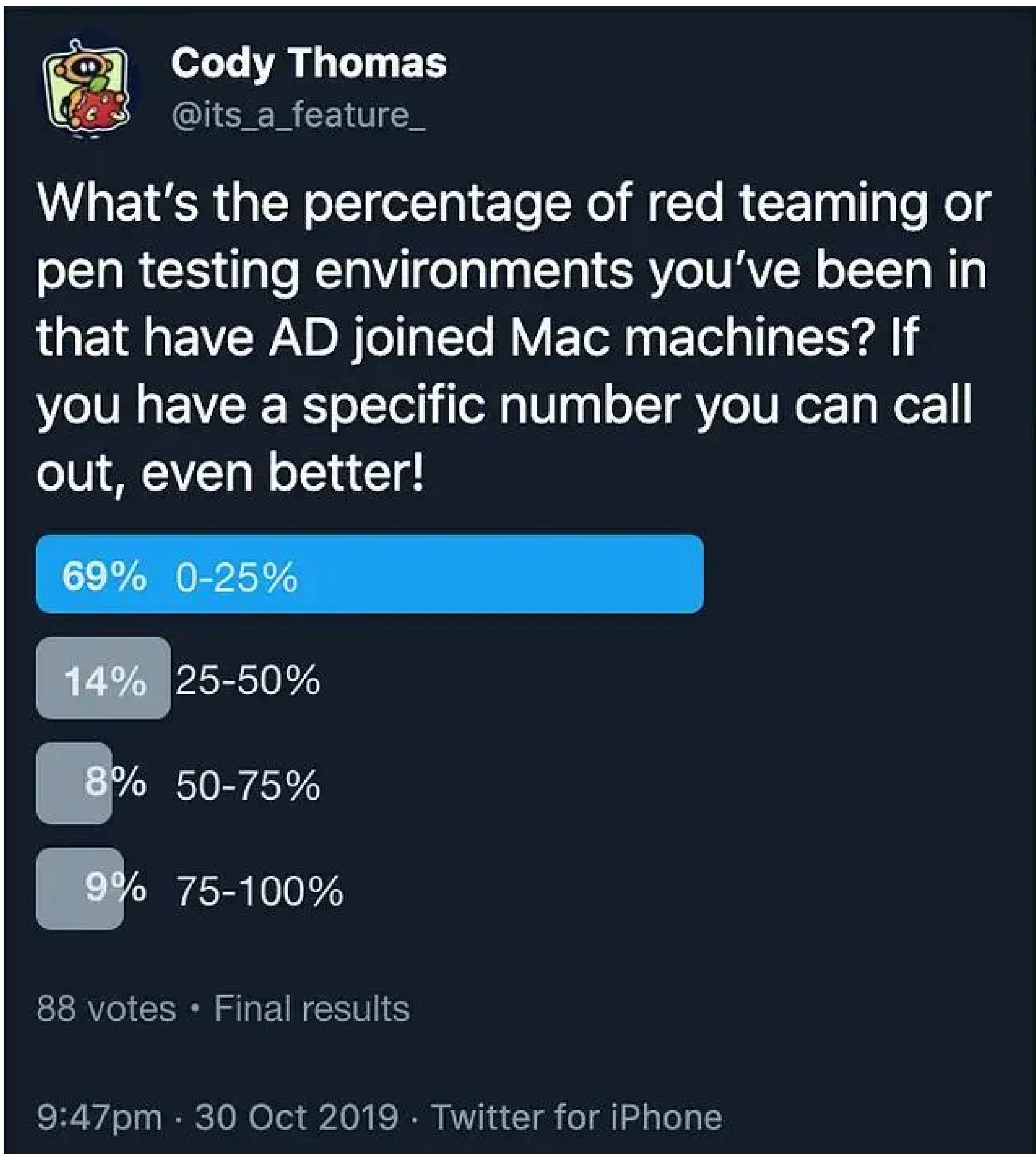
Published: 2019-11-14 · Archived: 2026-04-05 15:54:06 UTC

Red Teaming macOS

Apple moves forward in large jumps each time they release a new version of their Macintosh operating system. Each change can deprecate a whole suite of offensive tooling or add multiple extra hurdles. Because of this, red teamers and penetration testers historically gravitated towards things like Python and shell commands due to their stability. However, with the new direction Apple is headed, there's no guarantee for external languages like Python, Ruby, or Perl to exist on a macOS endpoint by default.

This pushes development of new capabilities lower into the operating system or into Apple-specific programming languages. After all, if Python isn't on the table, then neither are common tools like Empire, Impacket, or Responder. Even some of Apple's own versions of scripting languages like Javascript for Automation (JXA) are poorly maintained and not feature complete.

As a red teamer, assuming you manage to get execution on a macOS endpoint, two of the biggest pain points for assessments deal with credential access and lateral movement. One thing people often forget though is that credentials come in many varieties. There are more credentials on a macOS endpoint than just the user's password hash or plaintext password. Specifically, there are Kerberos tickets. This is more useful on an Active Directory (AD) joined computer, but how common is that really?



Twitter poll for frequency of seeing AD joined macOS endpoints in offensive engagements

From the quick poll results, at least 30% of people encounter an AD joined Mac at least 25% of the time. To help work with Kerberos tickets on macOS endpoints, I'm releasing a new, open source tool called Bifrost. Bifrost is an Objective C library that uses lower level Kerberos APIs and manual Kerberos network traffic to allow collection, manipulation, exfiltration, and discovery of Kerberos related information on macOS.

The rest of this post focuses on macOS specific background knowledge, abuses regarding Kerberos, and defensive considerations.

Active Directory is more than just Windows

When talking about Active Directory (AD), most people initially think about Microsoft, Windows, and sometimes Kerberos. Because of this, there are large amounts of research and tooling dedicated to the abuse and defense of these attack surfaces. Tools like Bloodhound seek to illustrate AD attack paths with graph theory, and tools

like Rubeus and Kekeo aim to highlight Kerberos specific abuses on Windows. This post will cover some of the same abuses such as over-pass-the-hash, kerberoasting, and pass-the-ticket, but from a macOS standpoint.

However, as enterprises grow, more systems besides Windows need to be managed, secured, and maintained. There are two main approaches to this problem: either join these new systems to the current AD environment, or find some other means of centrally managing them. While not necessarily an every-day scenario, macOS endpoints joined to a Windows Active Directory environment is becoming more common. Most endpoint security vendors on macOS are still looking for python instances, malicious command line binaries, or direct keychain access, which makes API access to Kerberos tickets very appealing from an offensive standpoint.

Kerberos Kerberos Kerberos

Since AD boils down to a management tool for organizing users, groups, policies, and containers, there needs to be auxiliary components that handle authentication and authorization. When it comes to network level authentication, Kerberos is the de facto standard; however, Kerberos is not a Microsoft construct.

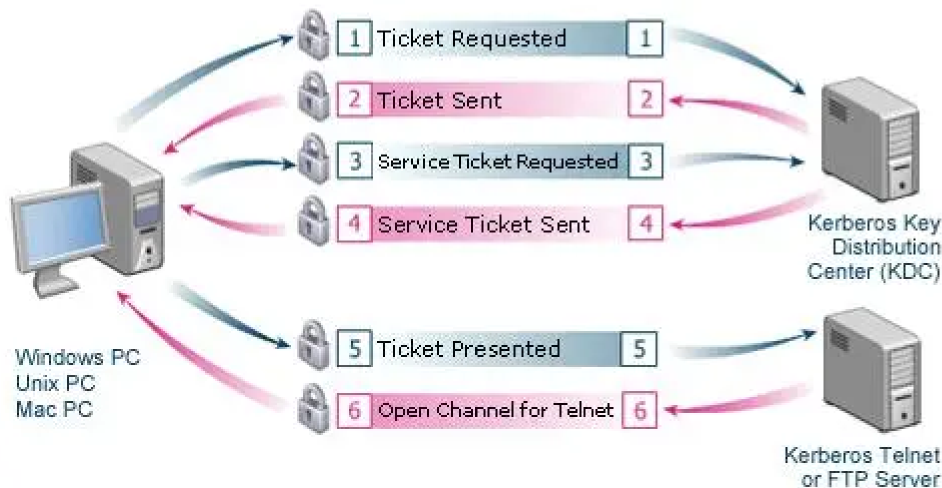
> Kerberos was created by MIT as a **solution to these network security problems**. The Kerberos protocol uses **strong cryptography** so that a client can prove its identity to a server (and vice versa) across an insecure network connection.

Because Kerberos is open source and freely provided by MIT, there are a few main variants in use today.

- The truest form is the actual MIT Kerberos implementation that’s used by various *nix applications.
- The most common implementation is the Microsoft adaptation that’s used in conjunction with Active Directory in Windows — this version is not open source.
- Heimdal Kerberos is another free and open source implementation of the Kerberos version 5 standard. This is the implementation included in macOS endpoints.

Generally, all of the frameworks built on the Kerberos version 5 specification have some degree of interoperability, but implementations and attack surfaces vary wildly. The rest of this blog will focus on the Heimdal implementation on macOS.

For those not familiar with Kerberos, the following image is a nice high level overview:



Kerberos in a nutshell

The process takes a user's password, hashes it, and uses that to get what's called a Ticket Granting Ticket (TGT) — steps 1 and 2. At this point, the user's plaintext password is not needed anymore. This TGT is the user's identity. When trying to access a network resource (file share, ssh, etc), the user presents this TGT to get a Service Ticket — steps 3 and 4. This is a ticket specific to the user and the service they want to access. The final step is to then provide this service ticket to the actual service to see if the user is authorized to access it, and if so, provide access (like remotely mounting a share) — steps 5 and 6.

The key here is that instead of needing a user's plaintext password or needing a user's hash to pass-the-hash, a user's TGT or service ticket are also valid ways to remotely authenticate as that user.

Active Directory Configuration

The first thing to discover from an offensive standpoint is the configuration of the AD joined macOS endpoint. Most times, there is a world readable file in `/etc/krb5.conf` that contains the configuration for interacting with the Active Directory environment. This is not a firm requirement though. A macOS computer can be joined to AD without this file, but it's very common. This file contains information such as:

- Supported encryption types for Ticket Granting Tickets (TGTs) and Service Tickets such as AES256, AES128, RC4-HMAC, and 3DES
- Allowed ticket flags such as forwardable, proxiable, and renewable
- Ticket lifetimes
- Default realms and Key Distribution Center (KDC) locations and ports
- Default ticket storage options

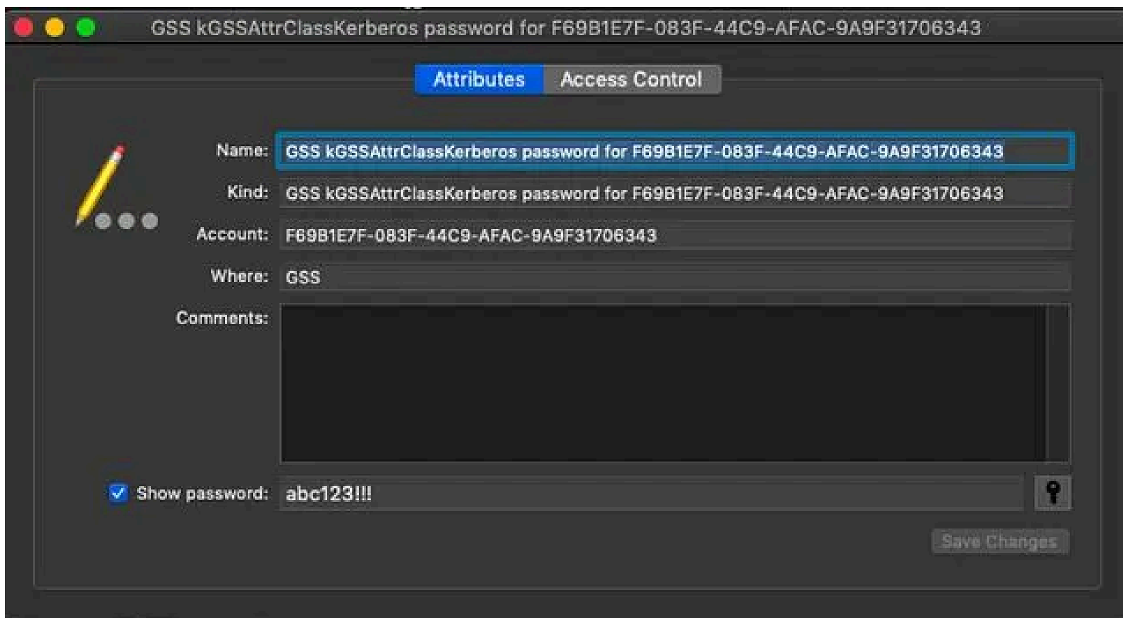
A user can also view the NETBIOS short-name for the domain they're connected to by going to System Preferences -> Users and Groups -> Login Options -> Network Account Server.

Ticket Storage on macOS

The next step is to identify where the Kerberos tickets are being stored. There are many different ways to store Kerberos credentials based on the OS and level of security desired, but by default, macOS uses Credential Cache (ccache) entries with a KCM — a process-based credential cache.

> Heimdal implements a credential cache type named "KCM" where operations are transmitted to a daemon process which manages the actual cache contents. On OS X 10.7 and later, the native default credential cache type uses the KCM protocol via Mach RPC. It is typically referred to via the "API" cache type for continuity with Kerberos for Macintosh; the API and KCM cache types have the same namespace in the native OS X Kerberos.

One way to view a limited subset of the credential cache is to open Keychain Access, click Keychain Access at the top and select Ticket Viewer. This is the only GUI version to view some of the ticket information for the current user. However, this is extremely limited. If Ticket Viewer is used to get and store Kerberos credentials, then this information is stored in the Keychain and can be revealed by providing the login keychain password. The UUIDs shown in the image below correspond to the credential cache names associated with the tickets:



Kerberos stored credentials in the keychain

Users normally interact with ticket storage and the macOS credential caches by using the following built-in binaries which do not save anything into the Ticket Viewer:

- kinit — gets a user's initial Ticket Granting Ticket (TGT) by taking in a plaintext password
- klist — lists some metadata about the entries in a specific ccache or lists out all of the visible ccaches in memory
- ktutil — works with keytab entries (more on that later)
- kcc — kerberos credential cache manipulations

From an offensive perspective, relying on built-in commands on the command line is an easy way to get caught. The Kerberos framework included in macOS by default exposes a bunch of lower level APIs for interacting with keytabs and ccaches. To get the similar functionality as `klist`, simply run the `list` action in Bifrost:

This loops through all of the discoverable in memory credential caches and displays information about the cache and all of the tickets inside (note: this does not need elevated permissions). The above example shows the `LAB\lab_admin` 's default cache (indicated with a *) has a type of `API`, meaning that the credentials are actually `KCM` credentials located in the kcm daemon. This cache has two entries — the user's TGT and an entry describing information to the KCM on how to access this data (krb5 vs gssapi). The first entry actually contains the user's TGT though, so that can be dumped with the `-action dump -source tickets` command:

There are a few things to notice about the above screenshot from dumping tickets. Bifrost will first describe the ticket it's about to dump. This includes information such as the principal, the encryption type used, when it expires, the flags present on the ticket, and the encryption key associated with the ticket. The next thing to notice is the giant base64 encoded blob labeled `Kirbi`. To facilitate operational usage of Bifrost, all tickets dumped and imported are in the Kirbi format. This is the same format used by Mimikatz, Kekeo, and Rubeus, among others. This allows tickets dumped from Bifrost to be immediately imported into a windows machine and vise versa.

Passing The Ticket

Importing tickets on macOS is analogous to importing tickets on Windows. The key to consider is that each credential cache is like a different logon session in windows. All of the credentials in a single credential cache are supposed to have the same client principal name. This would prevent having a TGT for Alice and a TGT for Bob in the same credential cache, but they could each have their own credential cache without conflicting. Let's assume you have a base64 encoded Kirbi ticket from another macOS machine or from a Windows machine and you want to import it. The following screenshot shows importing a CIFS service ticket to the Domain Controller into the default credential cache for the current user (note: this does not need elevated permissions).

If you're unsure about the contents of a Kirbi ticket and want to see what it actually contains, simply `-action describe` the ticket:

Over-Pass-The-Hash

A pretty common scenario in Windows is getting a user's hash, but needing to extend that into the Kerberos realm to get a full TGT instead. In similar fashion to Will (@harmj0y) 's Rubeus project and post, Bifrost can manually construct the ASN1 required Kerberos traffic to take user hashes and get TGTs back (note: this does not require elevation). If you collect an NTLM hash on a Windows machine, you should select the `rc4` encryption type.

Additionally, since the Kerberos traffic is being manually constructed, there are more options available. If you have the AES256 hash of the user's password, but want something more easily cracked (like RC4), then specify that with the `-tgtEnctype rc4` flag.

Kerberoasting

Now that you can get valid TGTs and import them, the next step is to use them to access services. Will covers Kerberoasting very thoroughly in his blog, so it won't be covered in-depth here, but this is something that can be achieved through Bifrost as well. The basic principal is that any valid TGT can be used to request a ticket to any service. The resulting ticket has a portion that's encrypted with the hash of password of the

associated service. So, if the service ticket is forced to use a weaker encryption, then it's easier to crack the service account's password.

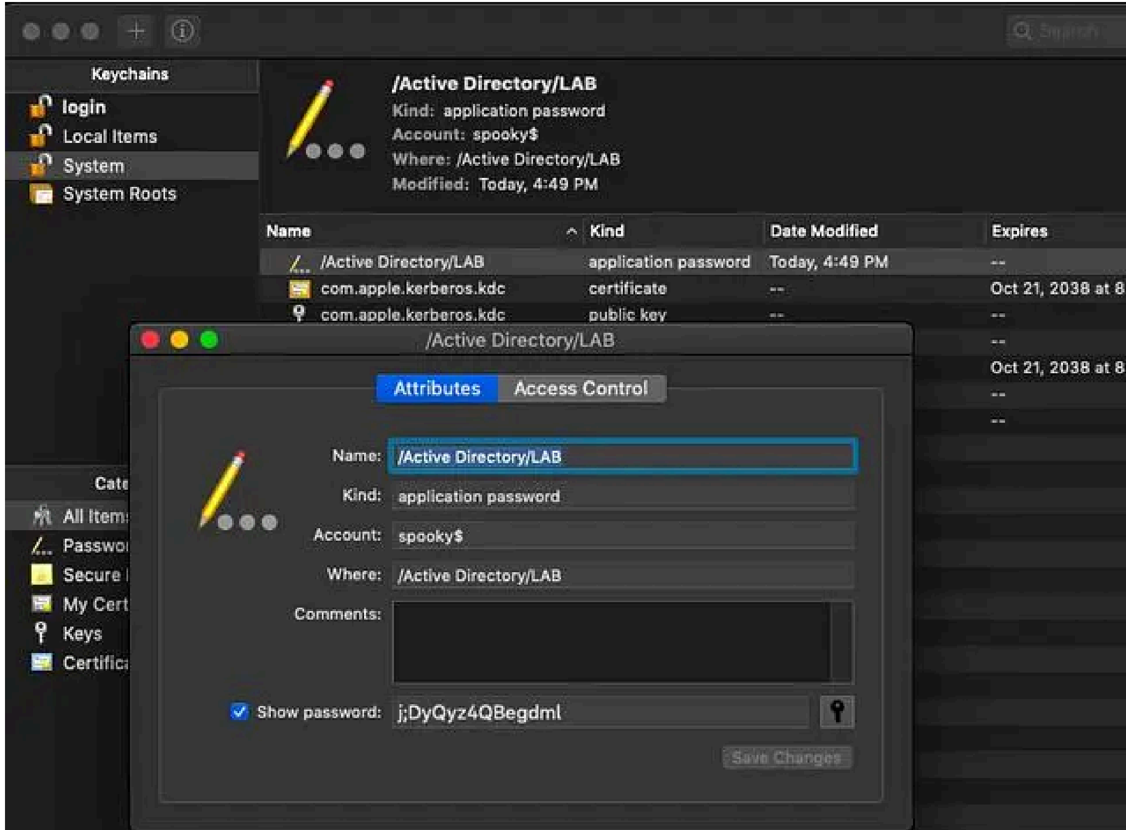
Since the only difference between getting a normal service ticket and kerberoasting is specifically requesting a weaker encryption type for the final ticket, the `asktgs` command in Bifrost simply takes an additional flag of `-kerberoast true` to request that the service ticket be RC4 encrypted instead of the standard AES256.

Multiple services can be specified at once by simply comma separating the SPNs. On macOS, there are four default SPNs set on the computer account:

- afpserver/spooky.lab.local
- cifs/spooky.lab.local
- host/spooky.lab.local
- vnc/spooky.lab.local

Computer$ Account

Acting as the local computer account to the domain is a little different in macOS than in Windows. In Windows, you can inject into a SYSTEM process and automatically act as the computer$ account remotely — this is why you can still query AD even though SYSTEM isn't a domain account. If you're able to unlock the system keychain (normally just need local admin credentials), then you can read the computer$ account plaintext password:



/Active Directory/LAB Computer$ account password access

Bifrost can use this password and the `-action askhash` command to generate the password hashes needed to get TGTs and service tickets for this computer. This password and other user passwords might have characters that tend to break or cause issues with command-line parsing, so if that's the case, simply use the `-bpassword` parameter instead and pass in the base64 version of the password as shown below:

```
spooky:~ lab_admin$ ./bifrost -action askhash -username spooky$ -domain lab.local -password "j;DyQyz4QBegdml"

Username: spooky$
Password: j;DyQyz4QBegdml
Domain: LAB.LOCAL
Salt: LAB.LOCALhostspooky.lab.local

Keys:
AES128: 1F44A5E5C7919C00F3166A1344D4FFDA
AES256: C1BF6861A00B35A97483E820863FAD4ED57831D935DBFE2D501727C678503F73
RC4   : A12AD40BD124E6A9A14D65504E8EA30A
spooky:~ lab_admin$ ./bifrost -action askhash -username spooky$ -domain lab.local -bpassword ajtEeVF6ejRRQmVnZG1s

Username: spooky$
Password: j;DyQyz4QBegdml
Domain: LAB.LOCAL
Salt: LAB.LOCALhostspooky.lab.local

Keys:
AES128: 1F44A5E5C7919C00F3166A1344D4FFDA
AES256: C1BF6861A00B35A97483E820863FAD4ED57831D935DBFE2D501727C678503F73
RC4   : A12AD40BD124E6A9A14D65504E8EA30A
```

Converting plaintext passwords into Kerberos-usable hashes

Leveraging Kerberos Tickets on macOS

After you've done some Kerberos ticket manipulation, how do you actually leverage it to access resources within the environment?

To access the SMB shares of a remote computer with your kerberos tickets, simply use mount:

```
mount -t smbfs "//computer/share" /local/path
```

It's important to remember that if a share is mounted, the local root user can access and traverse it. Any other users on the box can also run the `mount` command to view all the current mounts (similar to a `net use` on windows).

```
spooky:~ alice$ mount
/dev/disk1s1 on / (apfs, local, journaled)
devfs on /dev (devfs, local, nobrowse)
/dev/disk1s4 on /private/var/vm (apfs, local, noexec, journaled, noatime, nobrowse)
map -hosts on /net (autofs, nosuid, automounted, nobrowse)
map auto_home on /home (autofs, automounted, nobrowse)
map -fstab on /Network/Servers (autofs, automounted, nobrowse)
//lab_admin@dc1-lab.lab.local/C$ on /Users/lab_admin/mnt (smbfs, nodev, nosuid, mounted by lab_admin)
spooky:~ alice$
```

Alice seeing that lab_admin has a mounted share

You might run into issues with fully qualified domain names with the mount command, if that's the case, just the computer's name should work. The following video shows a normal user, Alice, taking the hash of the domain admin password to then mount and traverse the DC's C$ share:

Demo video of using over-pass-the-hash to access a C$ share

If you import the tickets into a credential cache other than the default (indicated with a *) or want to use credential caches on disk, you can specify `KRB5CCNAME=` as an environment parameter and specify the cache you want to use. If you're using one from memory, be sure to take the full name including the leading `API:`, and if you're using one on disk, be sure to prepend `FILE:` to the path.

SSH can also be configured to use Kerberos tickets, but that requires a lot more pieces to be in place for both the SSH client and the SSH server.

Keytabs and Saved Hashes

Keytab files are analogous to SSH keys. They store encryption information that's used to authenticate a user to a service. More specifically, keytab files contain the encryption keys used for Kerberos authentication and are tied to specific service principal names. The main point is that they can provide a mechanism for single-sign-on across a wide variety of applications and can be used on all major operating systems. While they are stored in plain text, they don't quite include the plaintext of passwords.

There is a default keytab file on every macOS device whether it's joined to a domain or not (more on that later) located at `/etc/krb5.keytab`, but users can create their own at any time. The main binary for working with keytabs is `ktutil`. This default keytab is locked down to the `root` user and the `_keytabusers` group on macOS with 640 permissions. When users create their own keytabs though, they can set different permissions on them.

```
spooky:~ alice$ ktutil -k test add -p lab_admin@LAB.LOCAL -V 3 -e aes256-cts-hmac-sha1-96
Password:
Verify password - Password:
spooky:~ alice$ ./bifrost -action dump -source keytab -path test
 ___          ___                  _
(  _`\  _   /'___)                ( )_
| (_) )(_)| (__  _ __   _     ___ | ,_)
|  _ <'| || ,__)( '__)/'_`\ /',__)| |
| (_) )| || |   | |  ( (_) )\__, \| |_
(____/'(_)(_)   (_)  `\___/'(____/\__)

[*] Resolving keytab path
[+] Successfully opened keytab
[+] principal: lab_admin@LAB.LOCAL
        Entry version: 3
        Key enctype: aes256
        Key: 2DE49D76499F89DEA6DFA62D0EA7FEDFD108EC52936740E2450786A92616D1E1
        Timestamp: 2019-11-10 04:58:09 GMT
spooky:~ alice$ ls -al test
-rw-------  1 alice  LAB\Domain Users  83 Nov  9 20:58 test
spooky:~ alice$
```

Creating and dump a new keytab

There are a couple important things to note here. Since this is a *nix environment, case sensitivity is important. Be sure to put the domain in all capitals or your hashes won't work. Additionally, each entry has exactly one key and

one encryption type, but there can be duplicate entries. If you noticed from above, the hash saved here is the same AES256 hash generated when using the plaintext password. Bifrost also supports specifying a keytab file and encryption type when requesting a TGT from the domain.

A Note on the LKDC

Every macOS device since 10.5 includes its own entire local kerberos stack — complete with unique realm, krbtgt user, and key distribution center. The Local Key Distribution Center (LKDC) was added to support Kerberos-based authentication between two Apple endpoints without requiring them to be joined to a central realm and KDC. This comes into play when accessing a shared service on a remote macOS endpoint (such as browsing the file system).

This process is a bit different than Windows-based Kerberos and is pretty apparent when you dump tickets with Bifrost after mounting a shared folder:

This blogpost won't go into all the details of the LKDC, how it works, or how it's different than the other AD joined Kerberos details in the rest of this blog (that'll be a future post), but there are a few "easy wins" from this data. The top highlighted area contains the `plaintext` password, username, and hostname of the remote machine. When dumping that ticket with Bifrost, you'll get a base64 version of the plaintext password used to mount the share. The bottom highlighted section shows where the share was mounted.

Keep in mind that this is an entirely different Kerberos realm with different methods of generating keys than the rest of this blog and different than Microsoft's implementation. Keys here can be found in the `/etc/krb5.keytab` file as well as in the `dsAttrTypeNative:HeimdalSRPKey` and `dsAttrTypeNative:KerberosKeys` attributes in local user's Open Directory properties. You can use the `dscl` utility or the Orchard project to read these, but you must be elevated to access any of them. The values stored here are in ASN1 notation, so if you want to explore then you should use an ASN1 decoder to parse out the pieces.

Defensive Considerations

There aren't a lot of native opportunities on macOS to delve into these APIs, but Heimdal Kerberos does include one way to at least enable debug-level logging on these API calls:

```
sudo defaults write /Library/Preferences/com.apple.Kerberos logging -dict-add krb5 '0-/SYSLOG:'
sudo defaults write /Library/Preferences/com.apple.Kerberos logging -dict-add kcm '0-/SYSLOG:'
sudo defaults write com.apple.MITKerberosShim EnableDebugging -bool true
sudo defaults write /Library/Preferences/com.apple.Kerberos logging -dict-add kcm '1-/ASL:'
```

By default, the `/Library/Preferences/com.apple.Kerberos.plist` file does not exist and is a world readable file when it is created. An attacker can read this file to determine potential logging capabilities and with elevation, disable it.

Additionally, with the new Endpoint Security Framework (ESF), it's possible to detect the `Kerberos.Framework` being loaded into processes in real-time. This will depend a lot of the specific

environment and which processes are normally leveraging this framework. Because Kerberos does offer a great way for Single-Sign-On (SSO) within an environment, there are likely to be a bunch of 3rd party applications that make use of these APIs.

Some of the Bifrost capabilities rely on doing manual Kerberos traffic to a domain controller (`asktgt` with a hash, `asktgs` , `s4u`). Because of this, there will be direct connections from the process running the Bifrost code to port 88 on the domain controller. Depending on the environment, this could potentially be an easy indicator.

Source: https://posts.specterops.io/when-kirbi-walks-the-bifrost-4c727807744f