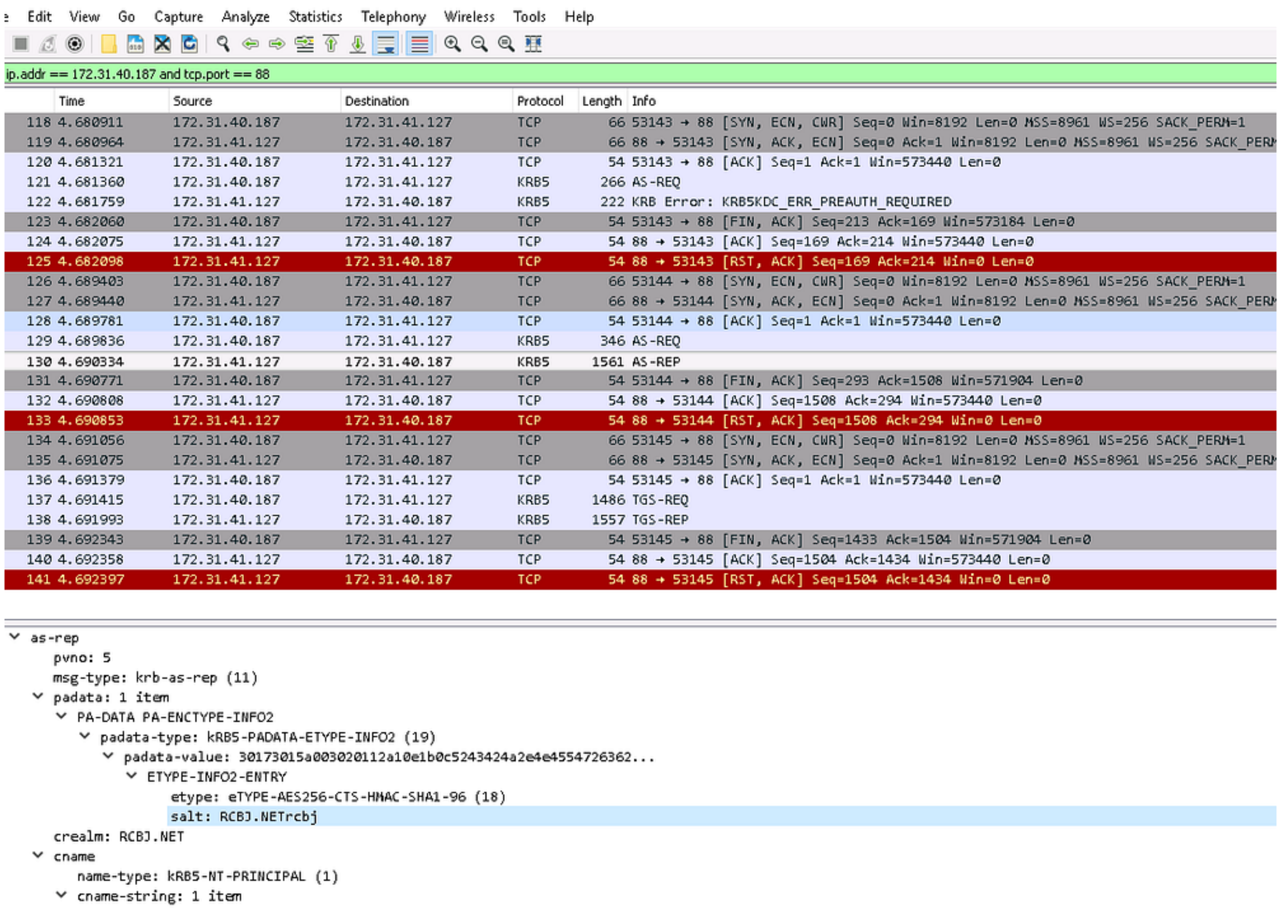
Kerberos Wireshark Captures: A Windows Login Example

medium.com/@robert.broeckelmann/kerberos-wireshark-captures-a-windows-login-example-151fabf3375a

Robert Broeckelmann

4 июня 2018 г.

Robert Broeckelmann



```
ip.addr == 172.31.40.187 and tcp.port == 88
```

```
      Time      Source          Destination     Protocol Length Info
  118 4.680911  172.31.40.187   172.31.41.127   TCP        66 53143 → 88 [SYN, ECN, CWR] Seq=0 Win=8192 Len=0 MSS=8961 WS=256 SACK_PERM=1
  119 4.680964  172.31.41.127   172.31.40.187   TCP        66 88 → 53143 [SYN, ACK, ECN] Seq=0 Ack=1 Win=8192 Len=0 MSS=8961 WS=256 SACK_PERM
  120 4.681321  172.31.40.187   172.31.41.127   TCP        54 53143 → 88 [ACK] Seq=1 Ack=1 Win=573440 Len=0
  121 4.681360  172.31.40.187   172.31.41.127   KRB5      266 AS-REQ
  122 4.681759  172.31.41.127   172.31.40.187   KRB5      222 KRB Error: KRB5KDC_ERR_PREAUTH_REQUIRED
  123 4.682060  172.31.40.187   172.31.41.127   TCP        54 53143 → 88 [FIN, ACK] Seq=213 Ack=169 Win=573184 Len=0
  124 4.682075  172.31.41.127   172.31.40.187   TCP        54 88 → 53143 [ACK] Seq=169 Ack=214 Win=573440 Len=0
  125 4.682098  172.31.41.127   172.31.40.187   TCP        54 88 → 53143 [RST, ACK] Seq=169 Ack=214 Win=0 Len=0
  126 4.689403  172.31.40.187   172.31.41.127   TCP        66 53144 → 88 [SYN, ECN, CWR] Seq=0 Win=8192 Len=0 MSS=8961 WS=256 SACK_PERM=1
  127 4.689440  172.31.41.127   172.31.40.187   TCP        66 88 → 53144 [SYN, ACK, ECN] Seq=0 Ack=1 Win=8192 Len=0 MSS=8961 WS=256 SACK_PERM
  128 4.689781  172.31.40.187   172.31.41.127   TCP        54 53144 → 88 [ACK] Seq=1 Ack=1 Win=573440 Len=0
  129 4.689836  172.31.40.187   172.31.41.127   KRB5      346 AS-REQ
  130 4.690334  172.31.41.127   172.31.40.187   KRB5     1561 AS-REP
  131 4.690771  172.31.40.187   172.31.41.127   TCP        54 53144 → 88 [FIN, ACK] Seq=293 Ack=1508 Win=571904 Len=0
  132 4.690808  172.31.41.127   172.31.40.187   TCP        54 88 → 53144 [ACK] Seq=1508 Ack=294 Win=573440 Len=0
  133 4.690853  172.31.41.127   172.31.40.187   TCP        54 88 → 53144 [RST, ACK] Seq=1508 Ack=294 Win=0 Len=0
  134 4.691056  172.31.40.187   172.31.41.127   TCP        66 53145 → 88 [SYN, ECN, CWR] Seq=0 Win=8192 Len=0 MSS=8961 WS=256 SACK_PERM=1
  135 4.691075  172.31.41.127   172.31.40.187   TCP        66 88 → 53145 [SYN, ACK, ECN] Seq=0 Ack=1 Win=8192 Len=0 MSS=8961 WS=256 SACK_PERM
  136 4.691379  172.31.40.187   172.31.41.127   TCP        54 53145 → 88 [ACK] Seq=1 Ack=1 Win=573440 Len=0
  137 4.691415  172.31.40.187   172.31.41.127   KRB5     1486 TGS-REQ
  138 4.691993  172.31.41.127   172.31.40.187   KRB5     1557 TGS-REP
  139 4.692343  172.31.40.187   172.31.41.127   TCP        54 53145 → 88 [FIN, ACK] Seq=1433 Ack=1504 Win=571904 Len=0
  140 4.692358  172.31.41.127   172.31.40.187   TCP        54 88 → 53145 [ACK] Seq=1504 Ack=1434 Win=573440 Len=0
  141 4.692397  172.31.41.127   172.31.40.187   TCP        54 88 → 53145 [RST, ACK] Seq=1504 Ack=1434 Win=0 Len=0
```

```
v as-rep
    pvno: 5
    msg-type: krb-as-rep (11)
  v padata: 1 item
    v PA-DATA PA-ENCTYPE-INFO2
      v padata-type: kRB5-PADATA-ETYPE-INFO2 (19)
        v padata-value: 30173015a003020112a10e1b0c5243424a2e4e4554726362...
          v ETYPE-INFO2-ENTRY
                etype: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
                salt: RCBJ.NETrcbj
    crealm: RCBJ.NET
  v cname
      name-type: kRB5-NT-PRINCIPAL (1)
    v cname-string: 1 item
```

RCBJ / Wireshark Screenshot

This blog post is the next in my Kerberos and Windows Security series. It describes the Kerberos network traffic captured during the sign on of a domain user to a domain-joined Windows Server 2016 instance. This post will help solidify our understanding of the Kerberos v5 protocol with a real world example. If you are new to the Kerberos protocol, a good starting place would be my Kerberos and Windows Security: Kerberos v5 Protocol post.

Luckily, the Kerberos protocol is mostly unencrypted (except for the tickets, authenticators, and some other sensative details) that rely upon message and field level encryption. This makes it easier to capture network traces (with Wireshark or similar tools) of Kerberos than some of the other identity protocols. Of course, many of the other identity protocols are built on top of HTTP(S) and tools like Chrome Developer Tools or similar can be used in the browser.

In this post, we will be using Wireshark v2.6.0. The traces were captured on the Windows Domain Controller that handled the Kerberos requests.

Assumptions

As always, we'll start with a bunch of assumptions to make sure we are in the same chapter (mostly given up trying to be on the same page).

- There is a lot going on with Kerberos in a Windows Domains. Some of it involves proprietary details beyond the scope of the Kerberos 5 protocol that we do not care about in this post. Here we are only interested in the pure-Kerberos details. See for a detailed description of Windows user login.
- I skip most of the details of what each actor is doing and instead focus on the messages exchanged by the protocol here. See this for more information about the processing done by each Kerberos actor.
- Likewise, I skip most of the introductory material in this post and jump straight to what is needed in order to understand the network traces. If you are having trouble following, please see this .
- The network traces captured for this post were generated with Windows Server 2016 running on AWS.
- No effort was made to obfuscate any of the information in these screenshots. All traffic was generated in a test environment that will no longer exist by the time this post is published.

Environment Setup

- Domain Controller was configured per instructions provided .
- DNS for internal rcbj.net domain was setup per instructions provided .
- A Windows Server was joined to the new domain per instructions provided .

Structure of a Kerberos Ticket

A Kerberos Ticket includes the following information:

Unencrypted Part:

- Version number of ticket format.
- Service realm
- Service principal

Encrypted Part:

- Ticket flags*
- Session key
- Client realm
- Client principal (username)

- List of Kerberos realms that took part in authenticating the user to whom this ticket was issued.
- Timestamp and other meta data about last initial request.
- Time client was authenticated.
- Validity period start time (optional).
- Validity period end time.
- Ticket Granting Server (TGS) Name/ID
- Timestamp
- Client (workstation) Address
- Lifetime
- Authorization-data — used to pass authorization data from the principal on whose behalf a ticket was issued to the application service (see of for more information)

*The following flags can be used in a ticket:

- reserved(0)
- forwardable(1)
- forwarded(2)
- proxiable(3)
- proxy(4)
- may-postdate(5)
- postdated(6)
- invalid(7)
- renewable(8)
- initial(9)
- pre-authent(10)
- hw-authent(11)
- transited-policy-checked(12)
- ok-as-delegate(13)

We will see two tickets in this example: Ticket Granting Ticket (TGT) and Service Ticket.

Structure of a Kerberos Authenticator

A Kerberos Authenticator contains the following information (all encrypted):

- Timestamp
- client ID
- application-specific checksum
- initial sequence number KRB_SAFE or KRB_PRIV messages)
- session sub-key (used in negotiations for a session key unique to this particular session)

Authenticators must not be re-used. A server that encounters a replayed authenticator must reject the message.

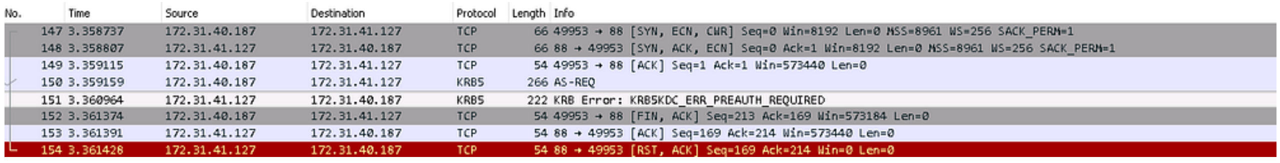
We will see one authenticator in this request: the authenticator sent with the TGT-REQ message.

Domain User Sign-On

The following screenshot shows the initial TCP connection between the domain-joined Windows server and the domain controller when the user “RCBJ\rcbj” tried to login via RDP. Note, the domain in question is called RCBJ (its DNS name is rcbj.net); the domain user we are tracing the sign in of is called rcbj. Yes, I like naming things after myself and it is duly noted that it is somewhat redundant in this case.

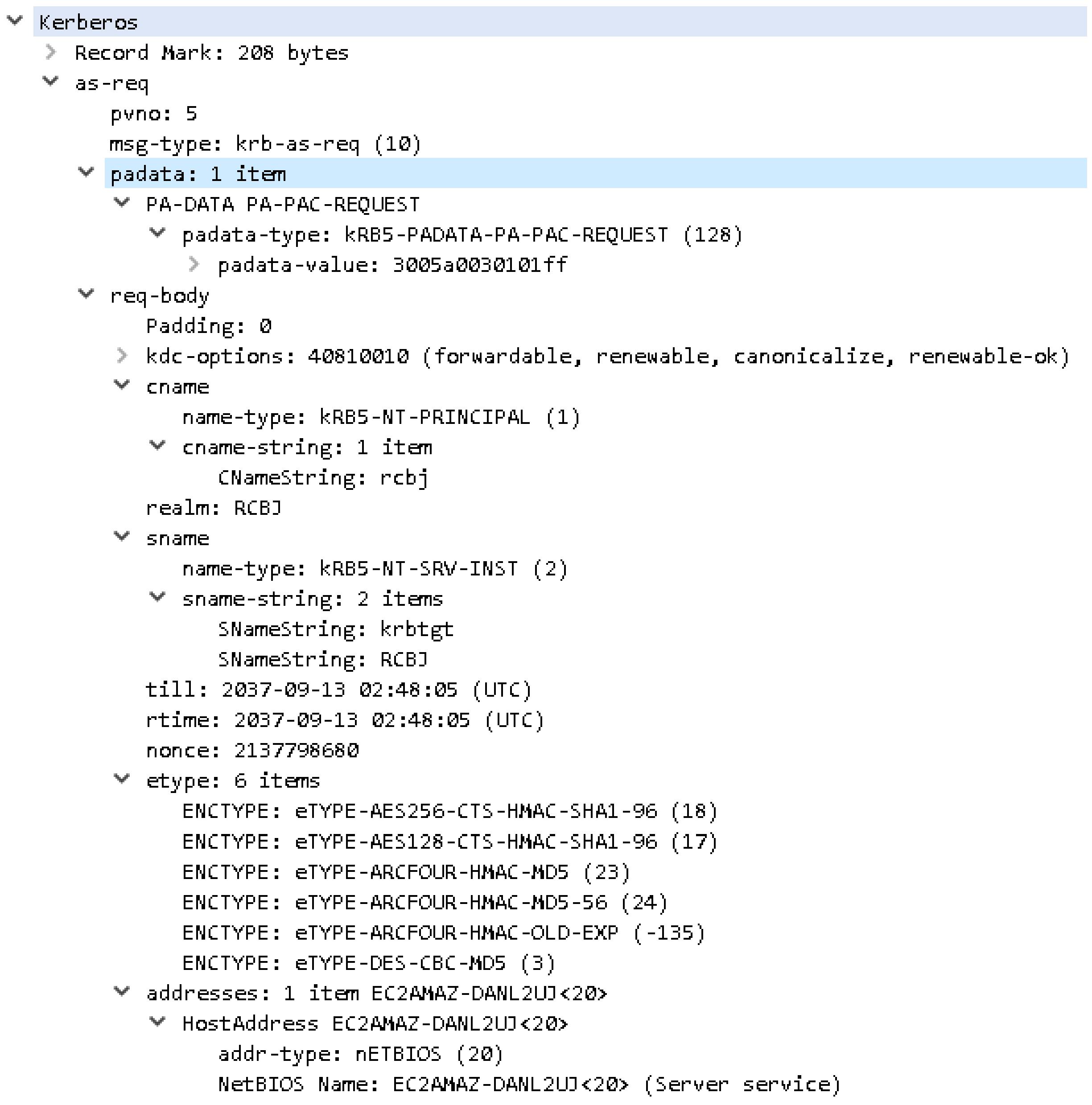
Authentication Service Exchange

The first three packets are the typical SYN-SYNACK-ACK handshake that happens for any TCP connection. This is one of my standard interview questions: What does the TCP handshake look like? 95% of all people I have ever interviewed couldn’t answer. The answer is quite simple.



No.	Time	Source	Destination	Protocol	Length	Info
147	3.358737	172.31.40.187	172.31.41.127	TCP	66	49953 → 88 [SYN, ECN, CWR] Seq=0 Win=8192 Len=0 MSS=8961 WS=256 SACK_PERM=1
148	3.358807	172.31.41.127	172.31.40.187	TCP	66	88 → 49953 [SYN, ACK, ECN] Seq=0 Ack=1 Win=8192 Len=0 MSS=8961 WS=256 SACK_PERM=1
149	3.359115	172.31.40.187	172.31.41.127	TCP	54	49953 → 88 [ACK] Seq=1 Ack=1 Win=573440 Len=0
150	3.359159	172.31.40.187	172.31.41.127	KRB5	266	AS-REQ
151	3.360964	172.31.41.127	172.31.40.187	KRB5	222	KRB Error: KRB5KDC_ERR_PREAUTH_REQUIRED
152	3.361374	172.31.40.187	172.31.41.127	TCP	54	49953 → 88 [FIN, ACK] Seq=213 Ack=169 Win=573184 Len=0
153	3.361391	172.31.41.127	172.31.40.187	TCP	54	88 → 49953 [ACK] Seq=169 Ack=214 Win=573440 Len=0
154	3.361428	172.31.41.127	172.31.40.187	TCP	54	88 → 49953 [RST, ACK] Seq=169 Ack=214 Win=0 Len=0

The forth packet that is sent from the windows server (Client) to the domain controller (KDC/Authentication Service) is the AS-REQ message (which is a type of KRB_KDC_REQ message). The AS-REQ message looks like the following:



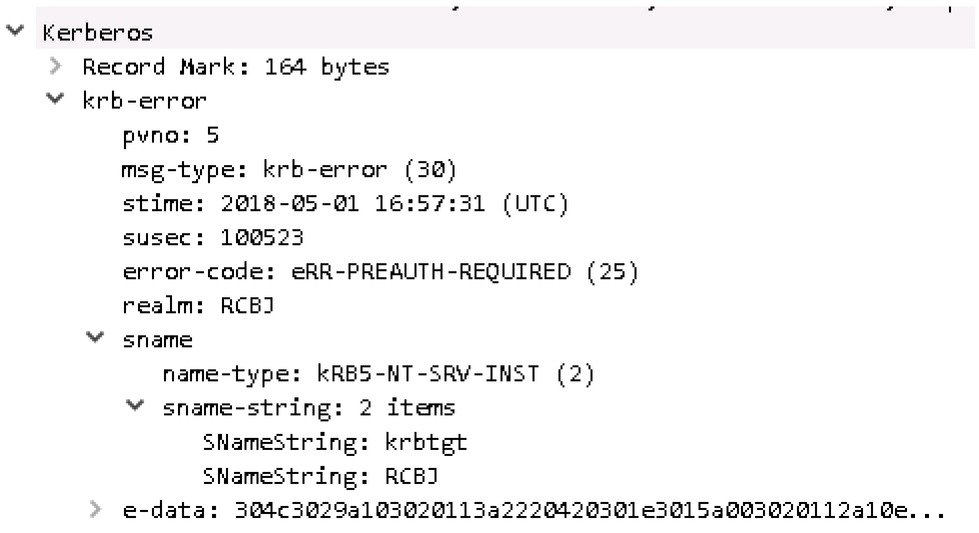
```
Kerberos
    Record Mark: 208 bytes
    as-req
        pvno: 5
        msg-type: krb-as-req (10)
        padata: 1 item
            PA-DATA PA-PAC-REQUEST
                padata-type: kRB5-PADATA-PA-PAC-REQUEST (128)
                    padata-value: 3005a0030101ff
        req-body
            Padding: 0
            kdc-options: 40810010 (forwardable, renewable, canonicalize, renewable-ok)
            cname
                name-type: kRB5-NT-PRINCIPAL (1)
                cname-string: 1 item
                    CNameString: rcbj
            realm: RCBJ
            sname
                name-type: kRB5-NT-SRV-INST (2)
                sname-string: 2 items
                    SNameString: krbtgt
                    SNameString: RCBJ
            till: 2037-09-13 02:48:05 (UTC)
            rtime: 2037-09-13 02:48:05 (UTC)
            nonce: 2137798680
            etype: 6 items
                ENCTYPE: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
                ENCTYPE: eTYPE-AES128-CTS-HMAC-SHA1-96 (17)
                ENCTYPE: eTYPE-ARCFOUR-HMAC-MD5 (23)
                ENCTYPE: eTYPE-ARCFOUR-HMAC-MD5-56 (24)
                ENCTYPE: eTYPE-ARCFOUR-HMAC-OLD-EXP (-135)
                ENCTYPE: eTYPE-DES-CBC-MD5 (3)
            addresses: 1 item EC2AMAZ-DANL2UJ<20>
                HostAddress EC2AMAZ-DANL2UJ<20>
                    addr-type: nETBIOS (20)
                    NetBIOS Name: EC2AMAZ-DANL2UJ<20> (Server service)
```

This is Message 0 (in my original protocol description). You can see the following fields in the screenshot above:

- pvno — Kerberos protocol version (5).
- msg-type — Application class tag number (10)
- padata — Pre-Authentication data that contains a structure. This is mentioned briefly in RFC4120, but goes into much more detail
- kdc-options — flags requested for the resulting ticket (forwardable, renewable, canonicalize, renewable-ok).
- cname — contains the user name being authenticated
- realm — contains the Kerberos realm (a.k.a. Windows domain name)
- sname — the service name being requested (in this case, it is again the windows domain name)
- till — The requested expiration time of the ticket being requested.
- rtime — If a renewable ticket was requested, this field contains the desired absolute expiration time for the ticket
- nonce — the message nonce

- etype — Requested encryption types
- addresses — the client IP address

The domain controller (authentication service) wants pre-authentication data to be provided. So, it returns the following Kerberos Error:

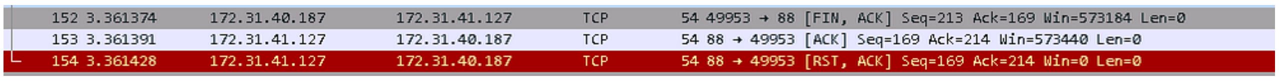


```
Kerberos
  Record Mark: 164 bytes
  krb-error
    pvno: 5
    msg-type: krb-error (30)
    stime: 2018-05-01 16:57:31 (UTC)
    susec: 100523
    error-code: eRR-PREAUTH-REQUIRED (25)
    realm: RCBJ
    sname
      name-type: kRB5-NT-SRV-INST (2)
      sname-string: 2 items
        SNameString: krbtgt
        SNameString: RCBJ
    e-data: 304c3029a103020113a2220420301e3015a003020112a10e...
```

Kerberos error messages are defined in RFC 4120, Section 5.9. The fields contained in this message are:

- pvno — Kerberos protocol version number (5)
- msg-type — Application class tag number(30)
- stime — Current time on the server at the time this message was generated.
- susec — This field contains the microsecond part of the server's timestamp.
- error-code — The error that occurred (pre-authentication data required, in this case)
- realm — the realm in which this error occured.
- sname — the service identifier (krbtgt@RCBJ, in this case, the Ticket Granting Ticket)
- e-data — additional data about the error for use by the application to help it recover from or handle the error.

At this point, the TCP connection is closed.



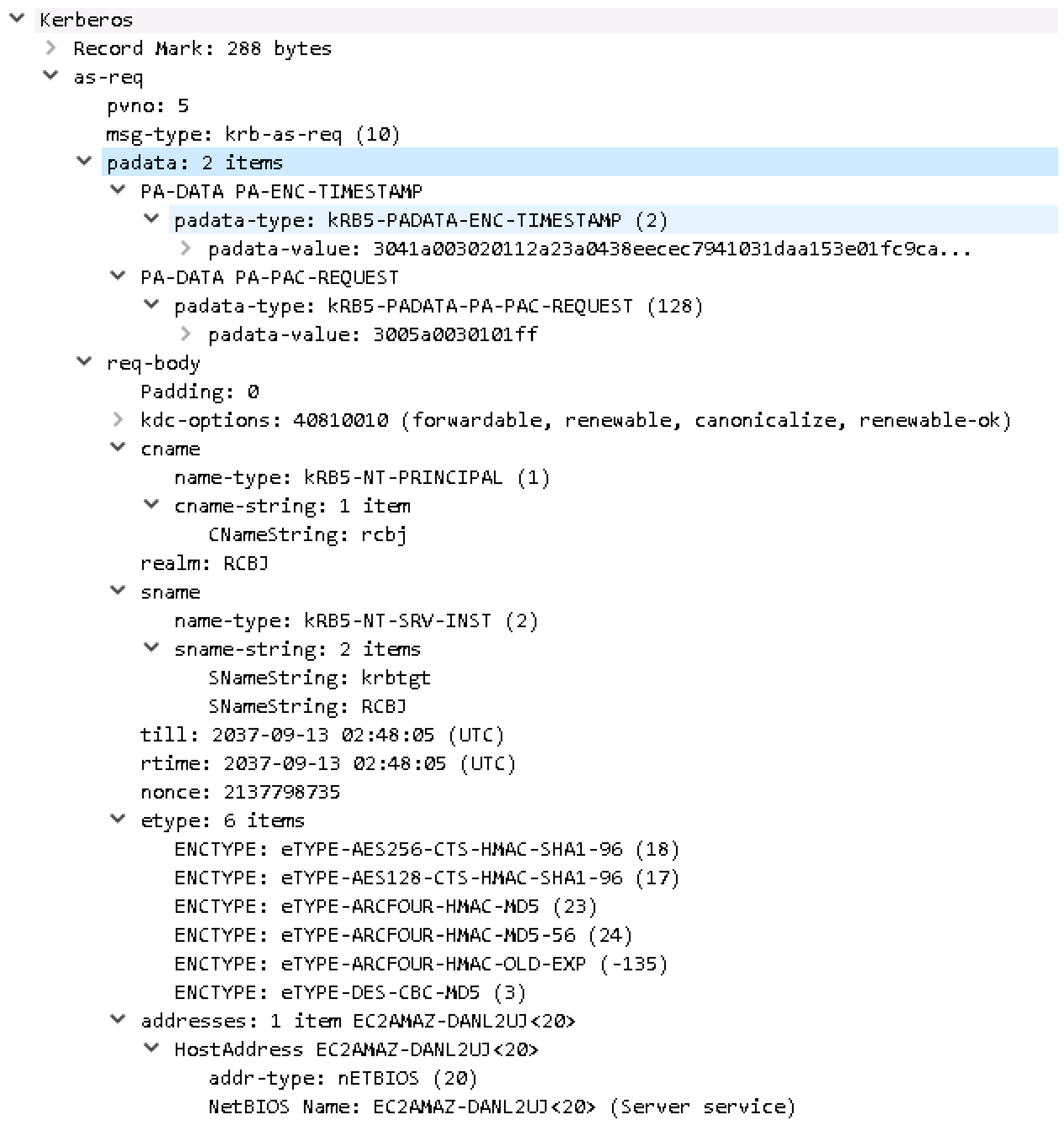
```
152 3.361374   172.31.40.187   172.31.41.127   TCP   54 49953 → 88 [FIN, ACK] Seq=213 Ack=169 Win=573184 Len=0
153 3.361391   172.31.41.127   172.31.40.187   TCP   54 88 → 49953 [ACK] Seq=169 Ack=214 Win=573440 Len=0
154 3.361428   172.31.41.127   172.31.40.187   TCP   54 88 → 49953 [RST, ACK] Seq=169 Ack=214 Win=0 Len=0
```

You can see the FIN, ACK, and RST (reset) packets that are exchanged as part of the standard connection termination.

Using the user's secret key derived from the windows password, a current timestamp is encrypted by the client (windows workstation or server) and used to populate the Pre-Authentication data with a KRB5-PADATA-ENC-TIMESTAMP message in the request below.

A new connection is established. I'm going to skip the connection stand-up and tear-down details this time around.

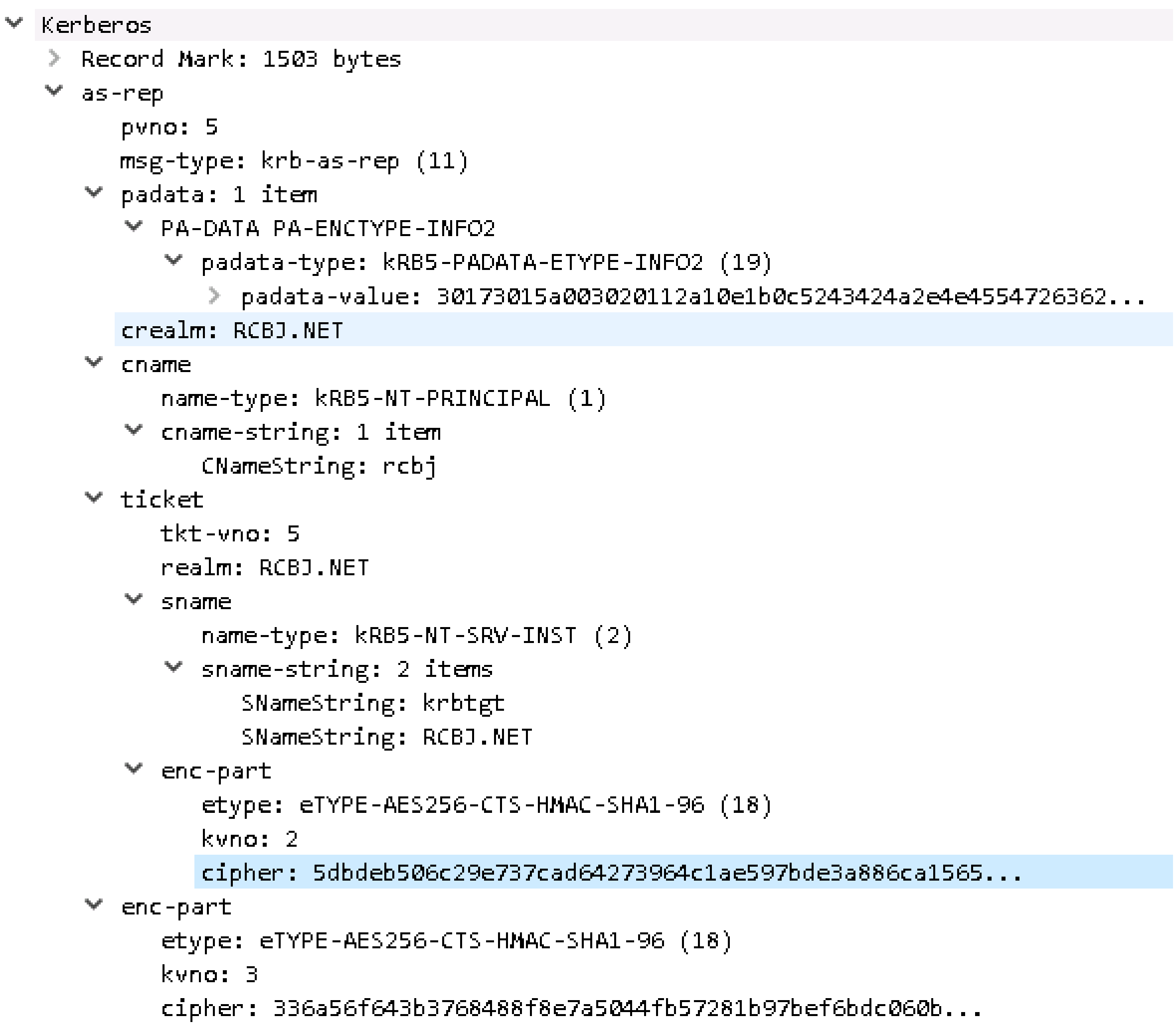
A new AS-REQ message is sent to the KDC Authentication Service:



```
Kerberos
  Record Mark: 288 bytes
  as-req
    pvno: 5
    msg-type: krb-as-req (10)
    padata: 2 items
      PA-DATA PA-ENC-TIMESTAMP
        padata-type: kRB5-PADATA-ENC-TIMESTAMP (2)
          padata-value: 3041a003020112a23a0438eecec7941031daa153e01fc9ca...
      PA-DATA PA-PAC-REQUEST
        padata-type: kRB5-PADATA-PA-PAC-REQUEST (128)
          padata-value: 3005a0030101ff
    req-body
      Padding: 0
      kdc-options: 40810010 (forwardable, renewable, canonicalize, renewable-ok)
      cname
        name-type: kRB5-NT-PRINCIPAL (1)
        cname-string: 1 item
          CNameString: rcbj
      realm: RCBJ
      sname
        name-type: kRB5-NT-SRV-INST (2)
        sname-string: 2 items
          SNameString: krbtgt
          SNameString: RCBJ
      till: 2037-09-13 02:48:05 (UTC)
      rtime: 2037-09-13 02:48:05 (UTC)
      nonce: 2137798735
      etype: 6 items
        ENCTYPE: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
        ENCTYPE: eTYPE-AES128-CTS-HMAC-SHA1-96 (17)
        ENCTYPE: eTYPE-ARCFOUR-HMAC-MD5 (23)
        ENCTYPE: eTYPE-ARCFOUR-HMAC-MD5-56 (24)
        ENCTYPE: eTYPE-ARCFOUR-HMAC-OLD-EXP (-135)
        ENCTYPE: eTYPE-DES-CBC-MD5 (3)
      addresses: 1 item EC2AMAZ-DANL2UJ<20>
        HostAddress EC2AMAZ-DANL2UJ<20>
          addr-type: nETBIOS (20)
          NetBIOS Name: EC2AMAZ-DANL2UJ<20> (Server service)
```

This is essentially the exact same message that came through the first time, but now the pre-authentication data is populated. I'm not going to rehash all the fields — see above.

After processing the request message, the authentication service returns the KRB_AS_REP to the client (workstation). It looks something like the following:



```
Kerberos
  Record Mark: 1503 bytes
  as-rep
    pvno: 5
    msg-type: krb-as-rep (11)
    padata: 1 item
      PA-DATA PA-ENCTYPE-INFO2
        padata-type: kRB5-PADATA-ETYPE-INFO2 (19)
          padata-value: 30173015a003020112a10e1b0c5243424a2e4e4554726362...
    crealm: RCBJ.NET
    cname
      name-type: kRB5-NT-PRINCIPAL (1)
      cname-string: 1 item
        CNameString: rcbj
    ticket
      tkt-vno: 5
      realm: RCBJ.NET
      sname
        name-type: kRB5-NT-SRV-INST (2)
        sname-string: 2 items
          SNameString: krbtgt
          SNameString: RCBJ.NET
      enc-part
        etype: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
        kvno: 2
        cipher: 5dbdeb506c29e737cad64273964c1ae597bde3a886ca1565...
    enc-part
      etype: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
      kvno: 3
      cipher: 336a56f643b3768488f8e7a5044fb57281b97bef6bdc060b...
```

The following fields are present in the KRB_AS_REP message:

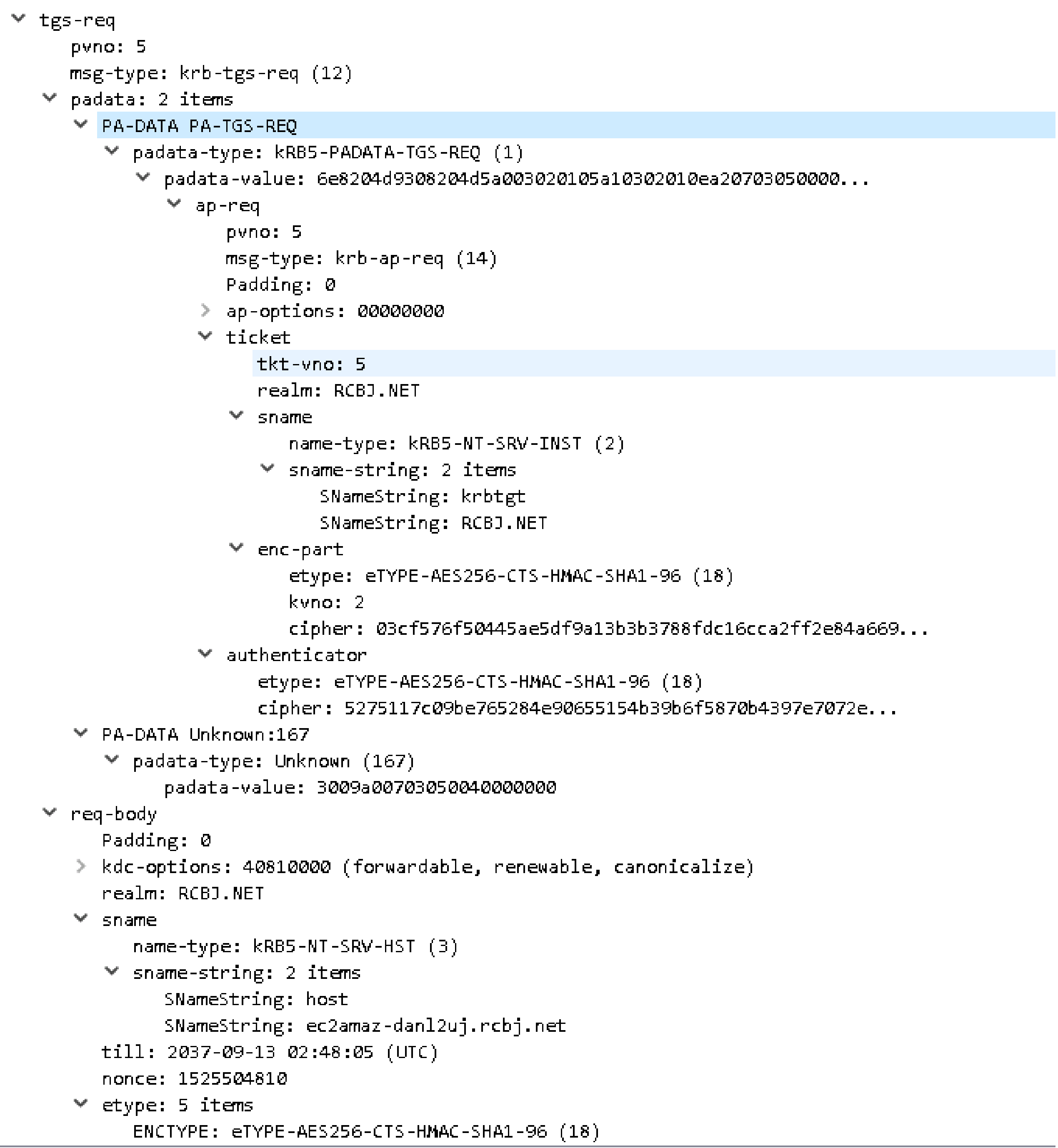
- pvno — The Kerberos protocol version number (5)
- msg-type — Application class tag number (10)
- padata — pre-authentication data (in this case, from RFC4120, Section 5.2.7.5, it will “provide information to the client about which key salt to use for the string-to-key to be used by the client to obtain the key for decrypting the encrypted part the AS-REP.”)
- crealm — The Kerberos realm (RCBJ.NET in this case)
- cname — The client/username (rcbj in this case).
- ticket — The Kerberos Ticket Granting Ticket for this session.
- ticket->tkt-vno —The ticket format version number (5).
- ticket->realm — The realm this ticket is issued for (RCBJ.NET in this case).
- ticket->sname — The service name this ticket belongs to, the KDC Ticket Graning Service. (krbtgt@RCBJ.NET)
- ticket->enc-part — The part of the ticket encrypted with the TGS’s secret key.
- enc-part — the client/TGS session key (encrypted with the user’s secret key)

These data fields represent Message 1 (the enc-part field that is encrypted with the user's secret, derived from the password) and Message 2 (the ticket field that contains the TGT). There is also some meta data included in the KRB_AS_REP message.

So, at this point, the client has a session key that it decrypted using the user's secret key and the TGT (which contains the same session key, among other things, encrypted with the TGS's secret key). In the next step, the TGS will have access to the client/TGS session key as well.

Token Service Exchange

Next, the client (workstation) establishes a new socket connection and sends the TGS-REQ message, which looks something similar to the following:



```
tgs-req
    pvno: 5
    msg-type: krb-tgs-req (12)
    padata: 2 items
        PA-DATA PA-TGS-REQ
            padata-type: kRB5-PADATA-TGS-REQ (1)
                padata-value: 6e8204d9308204d5a003020105a10302010ea20703050000...
                    ap-req
                        pvno: 5
                        msg-type: krb-ap-req (14)
                        Padding: 0
                        ap-options: 00000000
                        ticket
                            tkt-vno: 5
                            realm: RCBJ.NET
                            sname
                                name-type: kRB5-NT-SRV-INST (2)
                                sname-string: 2 items
                                    SNameString: krbtgt
                                    SNameString: RCBJ.NET
                            enc-part
                                etype: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
                                kvno: 2
                                cipher: 03cf576f50445ae5df9a13b3b3788fdc16cca2ff2e84a669...
                        authenticator
                            etype: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
                            cipher: 5275117c09be765284e90655154b39b6f5870b4397e7072e...
        PA-DATA Unknown:167
            padata-type: Unknown (167)
                padata-value: 3009a00703050040000000
    req-body
        Padding: 0
        kdc-options: 40810000 (forwardable, renewable, canonicalize)
        realm: RCBJ.NET
        sname
            name-type: kRB5-NT-SRV-HST (3)
            sname-string: 2 items
                SNameString: host
                SNameString: ec2amaz-danl2uj.rcbj.net
        till: 2037-09-13 02:48:05 (UTC)
        nonce: 1525504810
        etype: 5 items
            ENCTYPE: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
```

The TGT-REQ message's structure is similar the AS-REQ message we looked at earlier, which makes sense given that both of these messages have a common KRB_KDC_REQ definition (RFC 4120, Section 5.4.1). However, many of these fields are optional and missing here. RFC4120 states:

```
   The client prepares the KRB_TGS_REQ message, providing an
   authentication header as an element of the padata field, and
   including the same fields as used in the KRB_AS_REQ message along
   with several optional fields: the enc-authorization-data field
   for application server use and additional tickets required by
   some options.

   In preparing the authentication header, the client can select a    sub-session
key under which the response from the Kerberos server   will bemencrypted.  If the
client selects a sub-session key, care   must be taken to ensure the randomness of
the selected sub-   session key.
```

The TGT-REQ request contains:

- pvno: The Kerberos protocol version (5).
- msg-type: Application class tag number (12).
- pa-data: Pre-Authentication data field that contains an authentication header (see below).
- req-body: The request body.

In the sample message above, we can see the Pre-Authentication data field is populated with an authentication header that is of type PA-TGS-REQ (see RFC 4120, Section 5.2.7.1) data structure — it contains the TGT and authenticator for this step and is of type AP-REQ. The PA-TGS-REQ contains the following fields:

- pvno —Kerberos protocol version number
- msg-type — Application class tag number (14)
- padata — The Pre-Authentication data.
- crealm — The client realm name (Windows Domain name in this case).
- cname —The username (rcbj in this case).
- ticket — The ticket structure (TGT from Message 2)
- ticket->tkt-vno — The ticket format version number.
- ticket->realm —The realm the ticket was issued for.
- ticket->sname — The service this ticket was issued for (krbtgt@rcbj.net)
- ticket->enc-part —Encrypted part of the ticket.
- authenticator —Authenticator, encrypted using the Client/TGS Session Key

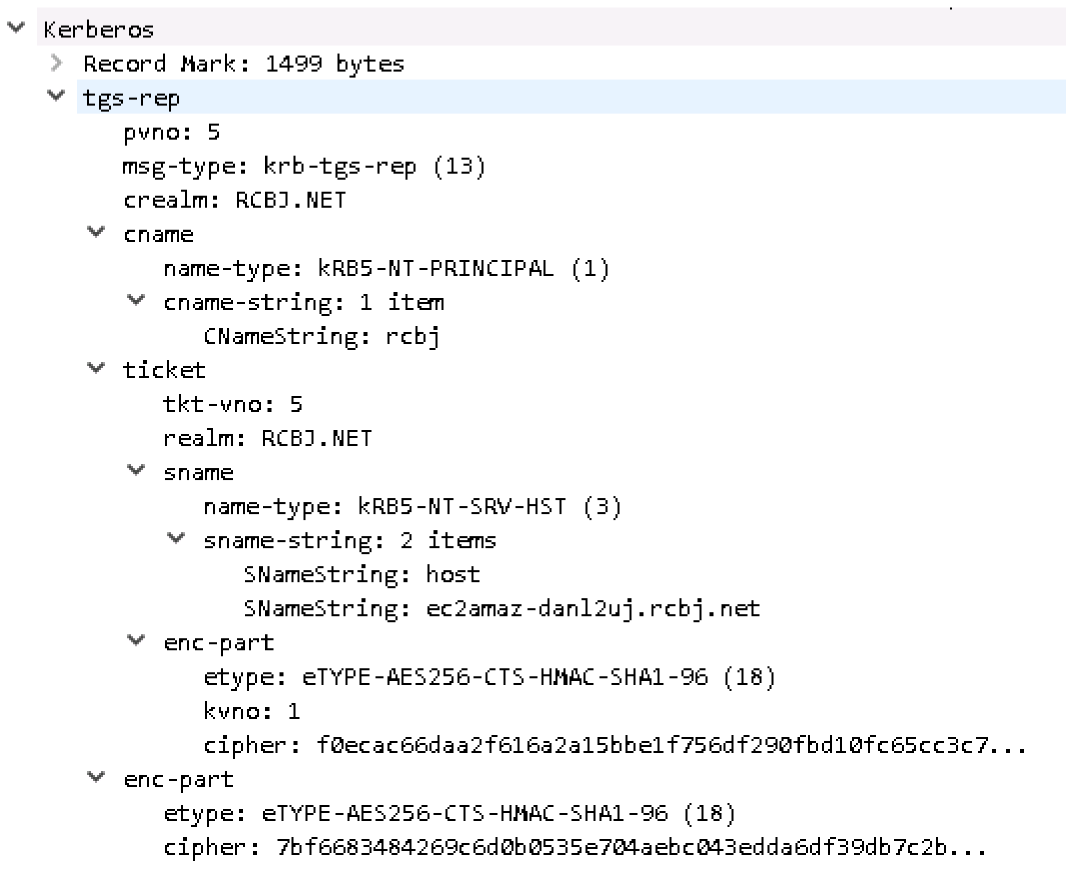
This includes Message 3 (the TGT from Message 2 and the ID of the requested service, krbtgt@rcbj.net for our Windows Domain login) and Message 4 (Authenticator, encrypted using the Client/TGS Session Key) from our earlier description.

At this point, the TGS uses its secret key to decrypt the TGT. Then, it can retrieve the Client/TGS session key from the decrypted ticket. Then, it can use the session key to decrypt the authenticator (enc-part field). The TGS can now compare the username in the authenticator to the name in the TGT. If the two names match, TGS proceeds as described below; otherwise, an error will be returned.

The TGS-REQ request body contains the following fields:

- flags: Flags describing what type of Service Ticket to return (in this case, the ticket should be renewable, forwardable, and canonicalized.
- realm: The realm the ticket is issued for (rcbj.net, in this case).
- sname: The service that the Service Ticket should be issued for (here, host@ec2amaz-danl2uj.rcbcj.net).
- till: Requested expiration time of ticket to be issued for this request.
- nonce: for this request.
- etype: The desired encryption algorithm(s) to be used in the response.

The KDC Ticket Granting Service responds back with a TGS-REP message:



```
Kerberos
  Record Mark: 1499 bytes
  tgs-rep
    pvno: 5
    msg-type: krb-tgs-rep (13)
    crealm: RCBJ.NET
    cname
      name-type: kRB5-NT-PRINCIPAL (1)
      cname-string: 1 item
        CNameString: rcbj
    ticket
      tkt-vno: 5
      realm: RCBJ.NET
      sname
        name-type: kRB5-NT-SRV-HST (3)
        sname-string: 2 items
          SNameString: host
          SNameString: ec2amaz-danl2uj.rcbj.net
      enc-part
        etype: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
        kvno: 1
        cipher: f0ecac66daa2f616a2a15bbe1f756df290fbd10fc65cc3c7...
    enc-part
      etype: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
      cipher: 7bf6683484269c6d0b0535e704aebc043edda6df39db7c2b...
```

The fields included are:

- pvno — The Kerberos protocol version number (5).
- msg-type — Application class tag number (13).
- crealm — The realm name (once again, the Windows Domain name,RCBJ.NET).
- cname — The username.

- ticket —The ticket structure (Client-to-server ticketencrypted using the service's secret key. The ticket is called a Service Ticket. In this example, the service is the local workstation's login service.
- ticket->tkt-vno — The ticket format version number.
- ticket->realm — The realm this ticket was issued for.
- ticket->sname — Service name this ticket was issued for (ec2amaz-danl2uj.rcbj.net, which is an ugly name, but what AWS auto-generated:).
- ticket->enc-part —Encrypted part of Client-to-Server (or Service) ticket.
- enc-part — Client/Server (or Service) Session Key encrypted with the Client-TGS Session Key.

This includes Message 5 (client-to-server or Service Ticket, encrypted using the service's secret key, or in this case, workstation's computer account's secret key) and Message 6 (client-server session key, encrypted with the client-TGS session key). To further clarify what secret key is being used to decrypt the Service Ticket in Message 5, recall that every workstation and server registered in the Windows Domain has a computer account defined that has a secret key (password) associated with it.

Upon receipt, the client can decrypt the client-server session key that it needs for the next step and the encrypted client-to-server ticket is available for submission to the desired service. Although, the Service Server named in the client-to-server ticket we have just obtained is the local server that we are logging into, which makes this particular example a bit odd in terms of the standard Kerberos use case — there is no remote server component that is being accessed. At least, not yet. Given enough time, the typical Windows user will do something that involves accessing a remote service (network file system, printer, email, etc). We'll see an example of the Client/Server Exchange a little later in this series.

Client/Server Exchange (or Lack Thereof)

During Windows login for domain users, the final exchange (KRB-AP-REQ and KRB-AP-REP messages) does not actually occur because the "service" we are hitting is the local workstation (or Windows server) login service, which is the same thing that initiated the Kerberos authentication sequence to start with. So, from the standpoint of having an example that demonstrates the Kerberos Protocol, this isn't the ideal example, but it is so common that it still seemed the best option for a first example in this series.

The first time I ever ran Wireshark to capture the examples used in this post, I spent about ten minutes looking for the third part, that was quite obviously missing. A few minutes of googling explained what I described above.

Summary

This is the first concrete Kerberos v5 example in this series. There will be more. Hopefully, this also helps anyone looking for an explanation of how Windows domain logins work.

Image: RCBJ / Wireshark Screenshot