Lessons in Disabling RC4 in Active Directory

syfuhs.net/lessons-in-disabling-rc4-in-active-directory

Steve Syfuhs March 2, 2021

Was pulled in to a fun customer issue last Friday around disabling RC4 in Active Directory. What happened was, as you can imagine, not good: RC4 was disabled and half their environment promptly started having a Very Bad Day.

— Steve Syfuhs (@SteveSyfuhs) March 1, 2021

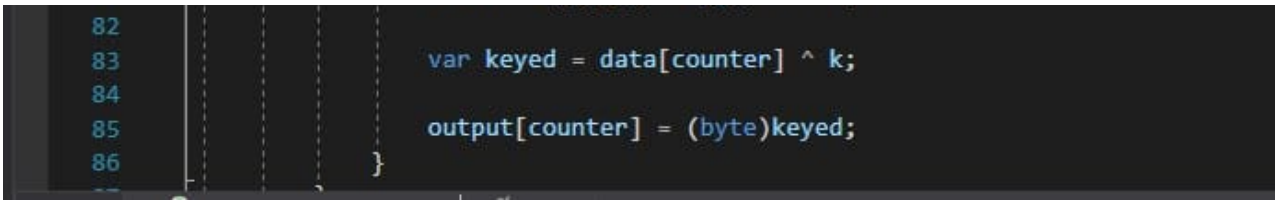
> **Twitter warning:** Like all good things this is mostly correct, with a few details fuzzier than others for reasons: a) details are hard on twitter; b) details are fudged for greater clarity; c) maybe I'm just dumb.

RC4 is a stream cipher. A stream cipher is kinda sorta like a one time pad (note: kinda, and sorta). A one-time pad is a cryptographic operation that takes one value and XORs it against a random value. A^B = C. A is your data, B is random noise. C is your encrypted cipher.

They're incredibly useful because the XOR operation is only reversible when you know A or B, and if B is suitably random that means you have to guess for all combinations of B. In other words you have to brute force it. As far as cryptography goes that's nearly perfection.

Kerberos.NET (netstandard2.0)

```
24              Span<byte> key = stackalloc byte[256];
25              Span<byte> s = stackalloc byte[256];
26
27              int i;
28
29              // for i from 0 to 255
30              //     Key[i]
31              //     S[i] := i
32              // endfor
33
34              for (i = 0; i < 256; i++)
35              {
36                  key[i] = originalKey[i % originalKey.Length];
37                  s[i] = (byte)i;
38              }
39
40              var j = 0;
41
42              // j := 0
43              // for i from 0 to 255
44              //     j := (j + S[i] + key[i mod keylength]) mod 256
45              //     swap values of S[i] and S[j]
46              // endfor
47
48              for (i = 0; i < 256; i++)
49              {
50                  j = (j + s[i] + key[i]) % 256;
51
52                  var swap = s[i];
53                  s[i] = s[j];
54                  s[j] = swap;
55              }
56
57              // i := 0
58              // j := 0
59              // while GeneratingOutput:
60              //     i := (i + 1) mod 256
61              //     j := (j + S[i]) mod 256
62              //     swap values of S[i] and S[j]
63              //     K := S[(S[i] + S[j]) mod 256]
64              //     output K
65              // endwhile
66
67              // E = data ^ k
68
69              i = 0;
70              j = 0;
71
72              for (var counter = 0; counter < data.Length; counter++)
73              {
74                  i = (i + 1) % 256;
75                  j = (j + s[i]) % 256;
76
77                  var swap = s[i];
78                  s[i] = s[j];
79                  s[j] = swap;
80
81                  var k = s[(s[i] + s[j]) % 256];
```



```
82
83              var keyed = data[counter] ^ k;
84
85              output[counter] = (byte)keyed;
86          }
```

However, the trick with one-time pads is that you need as many random bits as you have data. If you have 10 data you need 10 random. If you 10k data you need 10k random. You cannot repeat the random, lest you introduce a pattern and code breakers just love patterns.

So a stream cipher could take a one-time pad and cut the key down to a fixed length, manipulating the key every operation. Let's say you have 100 data and 10 random. The first 10 data get XOR'ed to the 10 random, then the 10 random get XOR'ed to something else. Repeat 10 times.

This turns out to be incredibly simple to code and is incredibly fast relative to other crypto algorithms. However, the cost is that you're now doing key scheduling which means if you can predict the schedule you've broken the cipher.

RC4 fits the bill here. It's painfully simple to implement. Here it is in entirety. But it's also irreparably broken.

The thing with RC4 is that if you have enough data transformed by a single key you can eventually predict what the original plaintext is. This became a semi-practical attack in 2013 when some smart folks figured out how to apply this to TLS. https://isg.rhul.ac.uk/tls/

For this attack to happen it requires observing billions of bytes of data. This is done easily enough with TLS, hence why folks jumped at disabling RC4 cipher suites. TLS isn't the only place RC4 is used, and RC4 is still broken, so it's just good form to disable it everywhere.

So now we have Active Directory and RC4 is enabled by default. In 2021?! How dare. Weeeeeellllll, RC4 isn't quite that bad in this case. Like, it's bad, but not super bad because each key will only ever be used for encrypting small amounts of data.

The current attacks require observing *lots* of data encrypted with a single RC4 key. For Kerberos we're talking maybe 8-16kb at most, not the gigabytes required for the attack. So it's bad, but not end of the world bad.

But that's no excuse. Why is it still there? Well, it turns out the RC4 cipher suite has a unique property: it doesn't require a salt when doing key agreement.

Huh?

So Kerberos has two legs between a client and KDC. AS and TGS.

AS is how you authenticate yourself: here's password gimme krbtgt.

TGS is: here's krbtgt gimme service ticket.

Now we don't ever just send the password to the server. What we do is we encrypt a thing using the password as a key and fire that thing over to the server. The server also has your password and can decrypt. If it decrypts then we agree we both know the password.

But Active Directory doesn't store the password itself. It stores a key derived from the password. That is, take the password and hash it, and store that hashed value. You encrypt against this hashed value.

So lets go back in time, circa mid 90's when Active Directory was being built. Back then, in the real world, Windows authentication was NTLM. NTLM kinda sorta worked similarly where you derived a key from a password and used that key to sign some stuff.

That derivation was and is admittedly very lame. It's `md4(password)`. So in NT Server in the directory database was username + md4(password), and that's it.

And taking the cryptographic properties of NTLM out the picture for a moment, it still kinda sucked as an authentication protocol. It didn't offer server authentication and it wasn't easily extendable for future changes. So Active Directory switched to Kerberos.

But we wanted upgrades to be seamless. When you left work Friday afternoon logging in to NT and came back Monday morning we wanted you to be able to log in to AD without any fuss, despite the fact that we changed literally everything out from under you.

And Kerberos at the time was using DES (eww). This posed a problem. You can't transform an MD4 key into a DES key. The protocol certainly didn't let you do weird things like DES(MD4(password)), so Windows created the MD4+RC4 crypto system.

The magic of this is that the NTLM key and Kerberos key are interchangeable so no password changes were required. Over time as users changed their passwords Active Directory would derive these newer, stronger, keys and would inform clients to prioritize these stronger keys.

This turns out to be phenomenally powerful because it transparently migrates users to stronger keys without breaking anyone, at a small cost of delaying weeks or months as password changes occur. Is it perfect? No. Does it keep the masses happy? Yes.

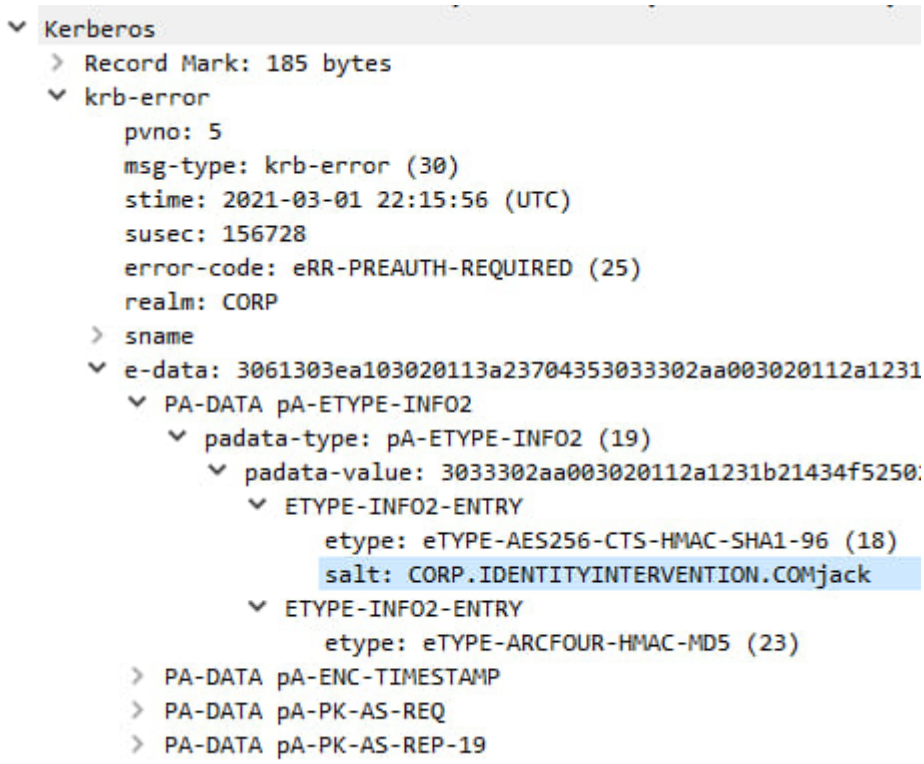
But there's another small problem: salts. Salts add randomness to passwords, making their hashed form incomparable to other hashes. hash('password', salt1) <> hash('password', salt2). We do this for all sorts of reasons, but the primary is so it's harder to guess the password.

> **Important**
>
> This lack of salt and the use of MD4 for password to key derivation is what makes the RC4 cipher suite in Kerberos dangerous. The RC4 portion itself is kinda meh in the overall scheme of things.
>
> This is because MD4 itself is a pretty lousy hash algorithm, and it's easier to guess the original password when compared to the AES ciphers. Passwords that are longer than 12 characters are generally safe from these kinds of attacks. I go into detail in Protecting Against Credential Theft in Windows.

But salts complicate the protocol. Both client and KDC need to take the password and salt and derive the same key, so both need to know them. Seems simple enough? Just use a well known value...?

Ah no, they need to be unique, otherwise two identical passwords with the same salt will form the same hash. Oops. Okay, compute a salt from the username. Aha, here we go, this works. It's unique and both parties know it. And it doesn't change! Oh wait, it can.

Users change their usernames all the time because names change all the time. If the salt were just the username that means we'd have to recompute the hash in AD every time the username changed, which means we'd need to know the password and only admins change usernames so 🤔.



Okay, so what if the KDC only changes the salt on password change and just tells us what it is whenever we authenticate? And so this is how Kerberos works. The client "pings" (no not ICMP) the KDC and the KDC says here ya go.

And so long story short (hahahaha, oh.) we come to the customer issue. RC4 doesn't require a salt. AES requires a salt. You can see in the previous screenshot the ARCFOUR instance doesn't provide a salt because there just isn't one.

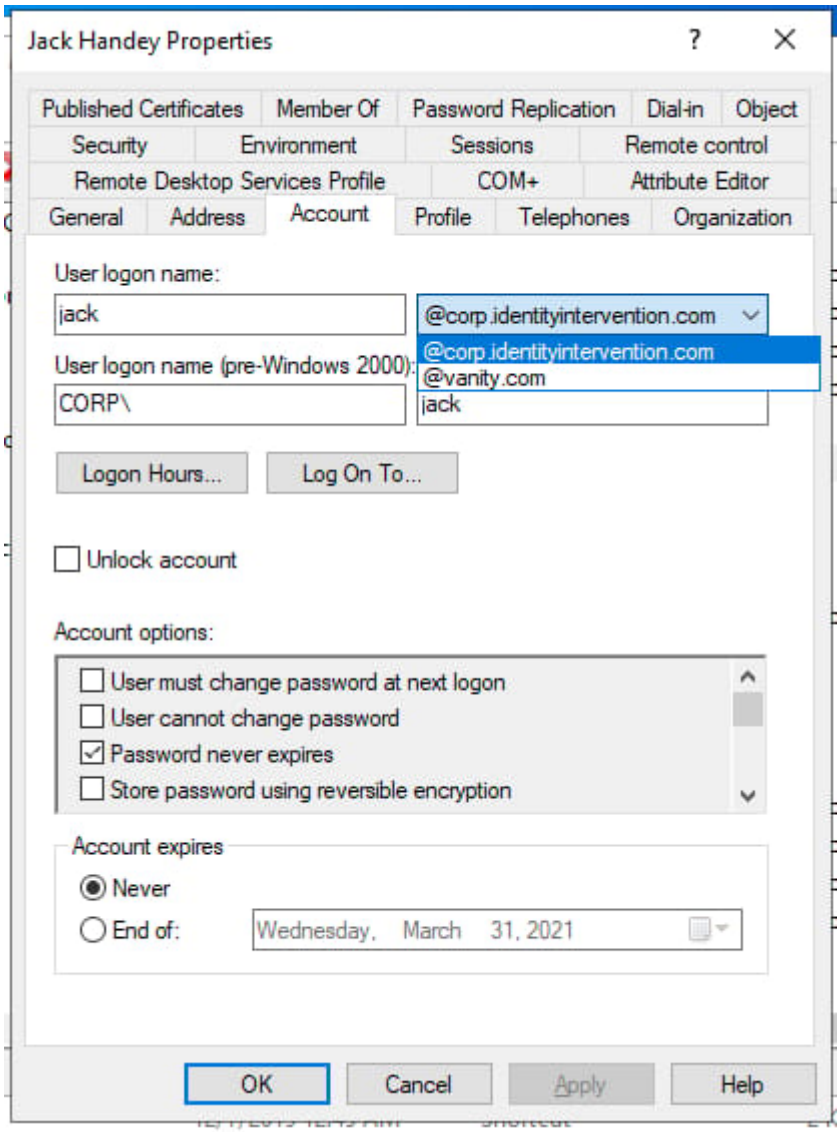
If you use RC4 you don't need to ping the KDC. You can just derive the key from the password and go. Therefore if you disable RC4 conversely you must always know the salt.

This poses a small challenge. Sometimes you don't want the system to know the password itself, so you provide these things called keytabs. They're files with some structured data that map users to derived keys. You derive the keys upfront once so you only need the password once.

Sometimes these files are generated without having line of sight to a KDC too so the tool generating the file has to guess which salt to use. Sometimes the tool guesses the wrong salt. Oops.

So you generate these files and deploy the thing and the system chugs along just fine until you disable RC4 and the whole things comes crashing down. The system was using RC4, and RC4 worked because there wasn't a salt. When it switched to AES it required the salt and...oops.

Well that's a pretty lame experience for anyone, let me tell you. Why did the tool guess the wrong salt? I've already told you: because the name changed.



Active Directory lets you change your username. You actually have two different usernames. Your sAMAccountName and your UPN. You can use either to log in with Kerberos, but Active Directory will only derive your salt from a single value: the sAMAccountName + Realm.

Well it happens that their UPN is in the form samaaccountname@customrealm.com, but their realm is anotherrealmentirely[.]com. As such their salt was ANOTHERREALMENTIRELY.COMsamaccountname. This happens when you add custom UPN suffixes.

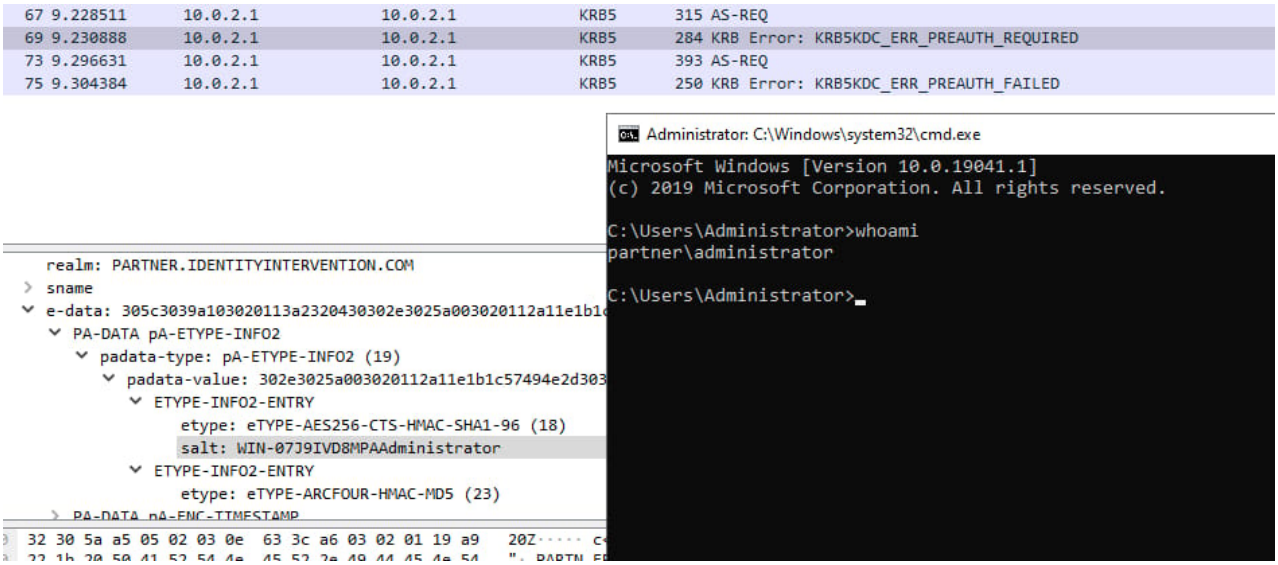
The password never changed so the salt never had a chance to change (not that it would). And the tool to generate the keytab just derived the salt from the user principal name, which in most cases is fine, but in this case wasn't.

And so you might think to yourself that this isn't that big a deal you haven't ever changed the usernames or the suffixes or anything like that. It turns out there's exactly one scenario that every environment hits: when you promote the first DC in a forest.

When you create a forest the domain admin is the machine's local administrator account. I'm not talking implicitly, I'm talking the local admin is copied into the AD database including it's password. The password that was set when you built the machine, before it had a realm.

And the domain admin is part of the protected users group so RC4 is already disabled for them. If the admin is a member of the Protected Users group, that means RC4 is disabled for them, and that then means they MUST use AES, which means they MUST use a salt, which means the salt had to exist when the machine was first built. Oy.

So the realm is the computer name.



Anyway. Here's some useful links as you consider getting rid of RC4.

Tough Questions Answered: Can I disable RC4 Etype for Kerberos on Windows 10 ? (microsoft.com)

Decrypting the Selection of Supported Kerberos Encryption Types - Microsoft Tech Community