Kerberos’ RC4-HMAC broken in practice: spoofing PACs with MD5 collisions

Two cryptographic flaws have been found in Kerberos as used by Microsoft Active Directory, which together can be exploited to bypass AD security features related to constrained delegation. One flaw is a forgery vulnerability in the legacy RC4-HMAC encryption type, that was supported by AD by default and rarely disabled. Due to a second flaw in the Privilege Attribute Certificate (PAC) protocol, which Windows uses to store authorization data in Kerberos tickets, this could be exploited in practice to bypass protections against constrained delegation attacks.

While the impact of the security feature bypass is somewhat limited, it does show that the RC4-HMAC encryption type is broken in a way that can lead to practical attacks. Since it is used in a variety of protocols, this opens up a new attack surface for all these protocols. This provides a strong argument in favor of moving towards retiring this encryption type.

Introduction to Kerberos

Kerberos is a protocol that allows members of a network to securely authenticate each other. Its most popular application is within Windows Active Directory, where it is used to authenticate individual domain users and allows them to prove their identity to domain services.

The basic idea behind Kerberos is that there is a trusted Key Distribution Center (KDC) which shares cryptographic keys with all members of the network. When a member A wishes to communicate to another member B, they contact the KDC to obtain a *ticket* and a *session key.* The ticket appears as an opaque binary string to A, but is actually encrypted and authenticated with the key shared between the KDC and B (but which is not known by A). Inside the encrypted ticket, there is a description of A’s identity and a copy of the session key.

Next, when A wants to authenticate to B, they submit this ticket along with a timestamped message encrypted with the session key. B can then unpack this ticket to determine the associated session key, and verify A’s identity by decrypting this timestamp. Depending on the protocol, they can continue using this key for post-authentication message encryption.

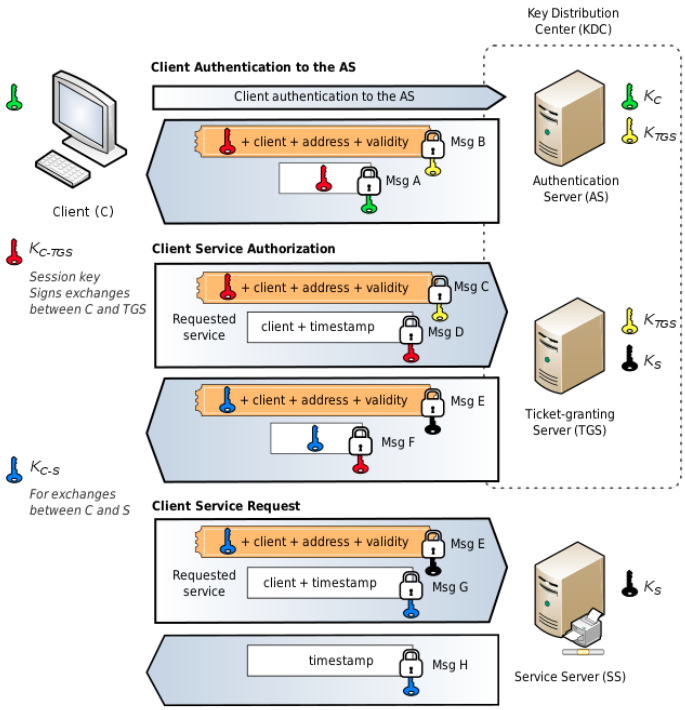

Figure 1: Illustration of the Kerberos protocol. Image source: Jeran Renz, Wikimedia Commons.

Kerberos has various *encryption types.* Each type define a set of cryptographic primitives that can be used to encapsulate tickets, verify identities or protect messages. A KDC can support multiple encryption types and will negotiate which one to use with each user that logs in. This whitepaper describes a vulnerability within a particular encryption type supported by Windows Active Directory by default, as well as a vulnerability in a Windows-specific Kerberos extension.

The problem with RC4-HMAC

Modern Windows systems support three Kerberos encryption types by default: *AES128_HMAC_SHA1*, *AES256_HMAC_SHA1* and *RC4_HMAC_MD5*. The AES-HMAC types are generally preferred, while the RC4-HMAC type has broader compatibility and uses the same shared secret key (derived by computing an unsalted MD4 hash of the user password) as the legacy NTLM authentication protocol.

The AES types use a separate key derived with a stronger password key derivation function. Therefore, enforcing AES for a user prevents overpass-the-hash attacks and makes it more difficult for attackers to brute-force passwords based on tickets collected with Kerberoasting and ASREPRoasting attacks.

Besides these known weaknesses, two components of the identifier RC4_HMAC_MD5 should also raise some red flags to those familiar with the cryptographic schemes RC4 and MD5, since both have well-known cryptographic vulnerabilities. RC4 is a stream cipher with statistical biases in its keystream that has led to practical attacks against TLS and other protocols. MD5 is a hash function that is not *collision-free*: an attacker can come up with two values that have the same MD5 hash.

Exploiting RC4 weaknesses in practice does however require an attacker to intercept a lot of data encrypted with the same key, which appears to be difficult to achieve in the case of Kerberos. Furthermore, when a hash function is used as part of the HMAC construction, which turns a hash function into a Message Authentication Code (MAC) scheme, collision resistance is not really necessary. As of 2022, no practical attacks against HMAC-MD5 have been found.

However, while HMAC-MD5 may be secure and RC4's weaknesses may be unexploitable, that does not mean that no mistakes can be made when composing these mechanisms in a cryptographic protocol. For this reason, I decided to take a look at RFC 4757, which defines how RC4-HMAC's cryptographic operations are defined. It didn't take long before I noticed a pretty clear problem:

```
The function is defined as follows:

K = the Key
T = the message type, encoded as a little-endian four-byte integer

CHKSUM(K, T, data)

     Ksign = HMAC(K, "signaturekey")  //includes zero octet at end
     tmp = MD5(concat(T, data))
     CHKSUM = HMAC(Ksign, tmp)
```

Figure 2: CHKSUM function as defined in RFC 4757.

This CHKSUM function produces a cryptographic MAC over some message and is supposed to be used by the Kerberos protocol and extensions thereof. It first uses HMAC to derives a new key for this purpose (that part is fine), then hashes the input and a type identifier with MD5, and then performs an HMAC of that MD5 hash.

While HMAC may be secure, HMAC'ing an insecure hash is not: if two inputs have a colliding MD5 hash this will result in the HMAC function being fed two identical inputs and thus producing the same output. This can be attacked as follows:

1. Given an attacker who has access to a *signing oracle* that computes a CHKSUM value over attacker input, but only if this input has specific properties that the oracle checks for.
2. The attacker produces two messages *data1* and *data2* such that MD5(T || data1) == MD5(T || data2), where data1 is a value that would be accepted by the signing oracle, while data2 is not.
3. The attacker submits data1 to the oracle, and obtains a checksum C.
4. Since C == CHKSUM(K, T, data1) == HMAC-MD5(Ksign, MD5(T || data1)) == HMAC-MD5(Ksign, MD5(T || data2)) == CHKSUM(K, T, data2), it holds that C is also a valid MAC over data2. Therefore, an attacker has obtained a MAC for a value not accepted by the signing oracle.

RFC 4757 also specifies three other operations. The second one is ENCRYPT, which is used to encapsulate Kerberos tickets and is therefore probably the most critical. Fortunately, this function applies HMAC-MD5 directly to the data it is encrypting, so it is not vulnerable to this attack.

The other two operations are GetMIC and WRAP. These are implementations of standard GSSAPI functions that can be used by other protocols to respectively authenticate or encrypt messages after

the Kerberos authentication handshake. These functions can be used within various protocols, such as LDAP and anything based on Microsoft RPC. These functions are used when an RC4-HMAC session key is negotiated during the Kerberos exchange, which is generally the case when either the server or client does not support AES ciphers.

Both `GetMIC` and `WRAP` are vulnerable to similar forgery attacks, since they both apply MD5 to their input before HMAC'ing it. This might cause vulnerabilities in any protocol using this functions, although a network MitM position is probably going to be necessary to exploit those.

Actually exploiting an MD5 collision

The forgery attacks against three out of the four functions defined in RFC 4757 are interesting in theory, but actually turning it into an exploit that achieves something useful for an attacker is quite tricky. The reason for this is that an attacker needs to find a MD5 collision between a " legitimate" payload that they can get authenticated by some signing oracle and a "useful" payload that makes the attacker achieve some malicious goal when they get use it in combination with a valid cryptographic MAC.

While it is trivial to compute some pair `(x,y)` for which it holds that `MD5(x) == MD5(y)`, there is no known method to compute `y` when `x` is a given value you can't control. So when you receive a MAC over some arbitrary message you can't generally change this message afterwards while keeping the MAC the same.

A powerful type collision that is practical to compute (although it takes some time) is a chosen-prefix collision: given two arbitrary prefixes `p1` and `p2`, an attacker can find a collision between `MD5(p1 || r1 || s)` and `MD5(p2 || r2 || s)` where `r1` and `r2` are sequences of random-looking bytes generated by the collision finding algorithm and `s` is a suffix that is fully under the control of the attacker. `r1` and `r2` are relatively long, but can be shortened at the cost of computing time.

To find something that might be exploitable with a chosen-prefix collision, I browsed around the documentation of various Kerberos and AD protocols looking for one that had a combination of the following properties:

1. It should use of one of the `CHKSUM`, `GetMIC` or `WRAP` functions.
2. There should be a method to make the protocol use the RC4-HMAC versions of these functions instead of the equivalents from the AES ciphers.
3. It should be possible to receive some message M1 authenticated by an automated process.
4. It should be possible to get a long, attacker-chosen, random string (r1) included in M1.
5. The bytes of M1 before the start of r1 (p1) should be completely predictable before the message is obtained.
6. It should be possible to construct a syntactically valid M2 that ends with a random string r2, directly followed by a suffix s, which equals the bytes from M1 past p1.
7. Having this M2 with a valid authentication tag should provide a useful advantage to an attacker.

After some reading and experimentation, I found one protocol that satisfied all of these requirements, which is described in the next section.

The PAC protocol

By default, a Kerberos ticket just contains the name and domain of the user authenticating to a service. There is also space for implementation-specific *authorization data* that provides additional information about the user's properties and privileges. Including this information in a ticket allows a service to make access control decisions without having to look up authorization information from a user repository.

In Windows AD environment, the Privilege Attribute Certificate (PAC) data structure is used for that purpose. A PAC contains various bits of information about the user's session, including security policy information and the AD groups the user is a member of. What drew my interest, however, was the inclusion of a two fields called "Server Signature" and "KDC Signature", which both include a cryptographic MAC that can be created using RC4-HMAC's vulnerable CHKSUM function:

SignatureType (4 bytes): A 32-bit unsigned integer value in little-endian format that defines the cryptographic system used to calculate the checksum. This MUST be one of the values defined in the following table. The corresponding sizes of the signatures are also given. The key used with the cryptographic system corresponds to the value of the **ulType** field of the outer PAC_INFO_BUFFER (section 2.4) structure. The value 0x00000006 specifies the server's key, and the value 0x00000007 specifies the KDC's key.

Value	Meaning
KERB_CHECKSUM_HMAC_MD5 0xFFFFFF76	As specified in [RFC4120] and [RFC4757] section 4. Signature size is 16 bytes. Decimal value is -138.
HMAC_SHA1_96_AES128 0x0000000F	As specified in [RFC3962] section 7. Signature size is 12 bytes. Decimal value is 15.
HMAC_SHA1_96_AES256 0x00000010	As specified in [RFC3962] section 7. Signature size is 12 bytes. Decimal value is 16.

Figure 3: Snippet from the [MS-PAC] protocol specification, showing supported ciphers for its signature feature. Image source: Microsoft Open Specifications.

These signatures are computed by a KDC (in Active Directory this is always a Domain Controller), while it is issuing a service ticket. Given a user U requesting a ticket for service S, the process is roughly as follows:

1. The KDC fills in all PAC fields, but fills the signature data with zeroes.
2. Using the service key (the same key, derived from the service user password, used to encrypt the service ticket), a cryptographic MAC is computed over the PAC. The result is written to the server signature field.
3. Using the KDC's own key (that of the special krbtgt account), a second MAC is computed over the server signature. The result, a MAC over a MAC, is written to the KDC signature field.

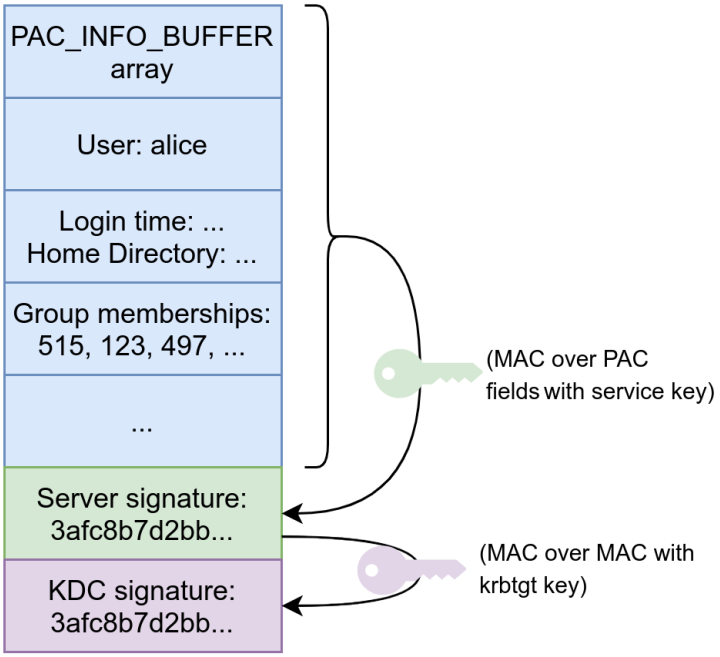

Figure 4: Illustration of PAC signatures. Note that the server signature covers the PAC contents, while the KDC signature just covers the server signature.

Note that during my research the specification was updated to include a third signature field, called a "ticket signature", with the purpose of binding the PAC stronger to the outer ticket in order to mitigate the "bronze bit" attack. This signature field is however not really relevant for the vulnerabilities described here, and when it is absent a PAC will still be accepted.

During typical usage of Kerberos, these signature fields are not that important: because the PAC is embedded in a ticket that is already encrypted with a service key an attacker will normally not be able to tamper with PAC contents. There are however a few situations when PAC tampering does become important: one of which is during constrained delegation.

PAC signatures and constrained delegation

There are many use cases where a service A wants to perform an action on some service B on behalf of a user U. For example: A could be a web-based file browser and B could be an SMB share of which U wants to access contents.

A could log in to B's share using its own service account. However, this would result in all users getting the same level of read/write access to files on the share, which may be undesirable. Furthermore, B will log reads and writes to the share as originating from the service account A, but the log will not actually make clear which end user is behind each interaction.

So ideally U would somehow give A permission to act on its behalf, and access B's share under U's account. One method to achieve this is through Kerberos *unconstrained delegation:* when A is given permission to use this feature, the KDC will embed a Ticket-Granting Ticket (TGT) in the service ticket that authenticates U to A. A can then unpack this TGT and use it to impersonate U when communicating with other services. This is risky, however, since A can not just impersonate U towards B, but also to any other service within the domain. If an attacker were to compromise A,

they would be able to abuse this position to automatically and fully take over the user accounts of anyone logging in to the file browser website.

There is also a more fine-grained mechanism to allow this kind of access: namely by allowing *constrained delegation* between A and B. By configuring a constrained delegation relation between A and B within a domain, you are basically saying that “A is allowed to impersonate users when talking to B”. However, this does not necessarily imply that A is allowed to impersonate users to some other service C, or that B is allowed to impersonate towards A.

This mechanism is facilitated through the *S4U2proxy* protocol: when server A possesses a service ticket from U to itself it will be able to present this to the KDC in exchange for a service ticket from U to B, as long as a constrained delegation relation exists between A and B and the account U is allowed to be delegated. A can then use this ticket to authenticate to B with the name and privileges of U.

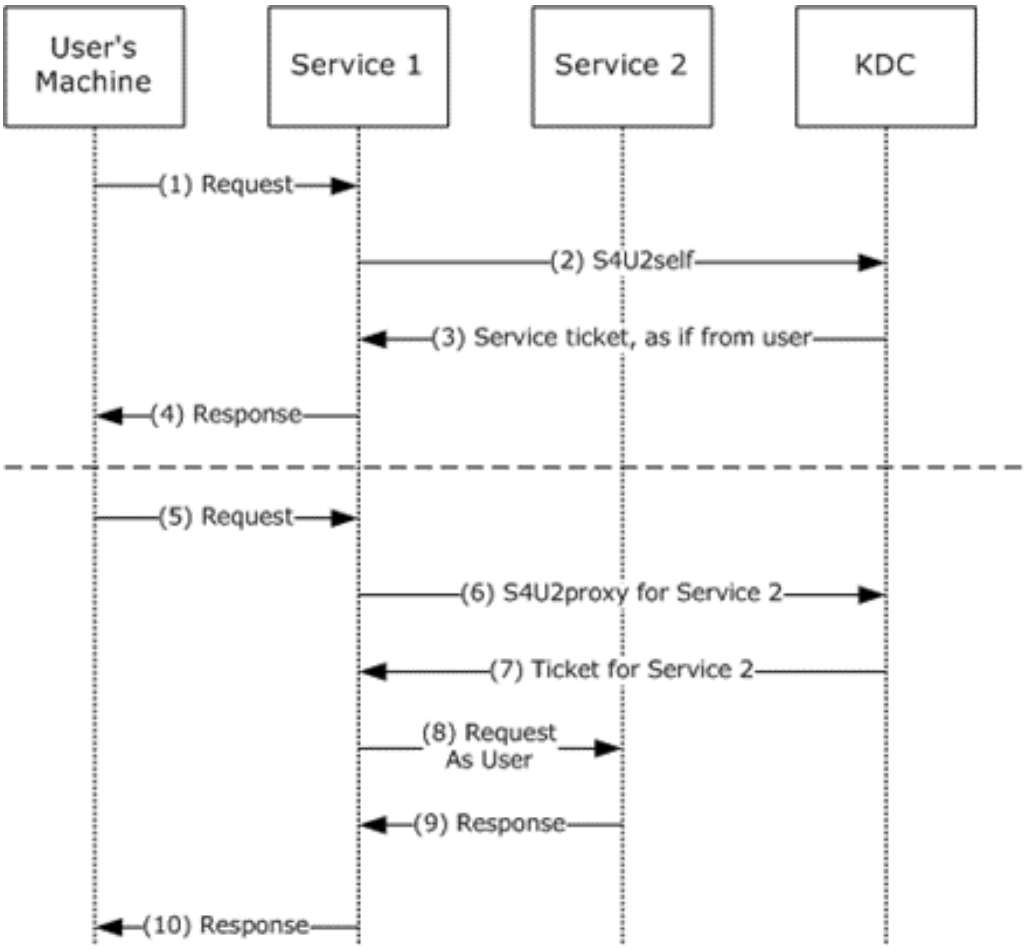

Figure 5: Illustration of constrained delegation process. First a machine requests a ticket from a user to itself with the S4U2Self protocol. Then it exchanges that for a service ticket impersonating that user with the KDC. Image source: Microsoft Open Specifications.

This protocol creates a problem, however, since the U-to-A ticket will be encrypted with a key known to A. This means that A is able to decrypt this ticket, make alterations to the PAC (for example making it say that U has more privileges than they actually possess) and then re-encrypt the ticket again. To prevent this from happening the KDC will verify the server and KDC signatures, and reject the ticket when these are not correct.

Since the server signature is set with a key know to A it can be updated by A to authenticate a maliciously altered PAC. When the server signature changes the KDC signature will no longer be valid, and since A will not know the krbtgt key it should not be able to update the KDC signature accordingly.

However, this protection is only effective under the assumption that altering the PAC will invalidate the server signature. If A could change the PAC in such a way that the original server signature remains valid, then the same KDC signature will also be valid. It turns out that the RC4-HMAC vulnerability makes this possible.

Spoofing a PAC with an MD5 collision

If an attacker can get the KDC to fill in the signature fields of a PAC X that has an identical MD5 hash to some other PAC Y, then the attacker can attach those signature fields to Y and make it valid. In order to achieve this in practice, the attacker must smuggle in a large string of random-looking bytes into both X and Y to make the collision work. Furthermore, they must be able to make the KDC produce a signed version of X that includes these random bytes.

The owner of a user account can not generally change much about their own PAC. However, it turns out that when a user joins a new computer account to the domain, they will then get control over various account properties for that computer account that appear in the PAC. Since by default each AD user is permitted to join 10 computers, it is usually easy to add a new attacker-controlled computer account that can be modified in this way.

Unfortunately there are no controllable fields in a PAC that can contain arbitrary byte strings, but one field that comes close is “LogonScript”, which contains the value of the LDAP “scriptPath” property (normally used to specify a path for a script to automatically run after logging in with an account; I don’t know if it has any meaning for a computer account though). This field is allowed to be quite long and can contain arbitrary UTF-16 strings.

Now, an MD5 collision algorithm is not normally going to output valid UTF-16. Luckily, Windows is pretty liberal on which Unicode code points it accepts, and the only strings it rejects are those with invalid surrogate pairs. A surrogate pair consists of two 16-bit words in between 0xD800 and 0xDFFF. If an even-sized byte string does not contain any 16-bit word that falls into this range, Windows will accept it a legal value for LogonScript/scriptPath.

I used Marc Stevens’ HashClash tool to compute the chosen-prefix collision. This tool is non-deterministic and operates in several *steps*: every step, a collision ‘block’ is emitted that consists of 64 bytes. The bits in these blocks are roughly randomly distributed, resulting in a chance of around 1 in 3 that a block happens to be valid UTF-16. I tweaked the tool to check the block after each step, and retry the last step if it contains a 16-bit word in the range 0xD800 and 0xDFFF. This allowed me to find an UTF-16-compatible MD5 collision in about three times the time it would take to find a general collision. On average, this took about 7 hours on the hardware I used.

Now all the ingredients for PAC spoofing were present, allowing for the attack shown in the figure below:

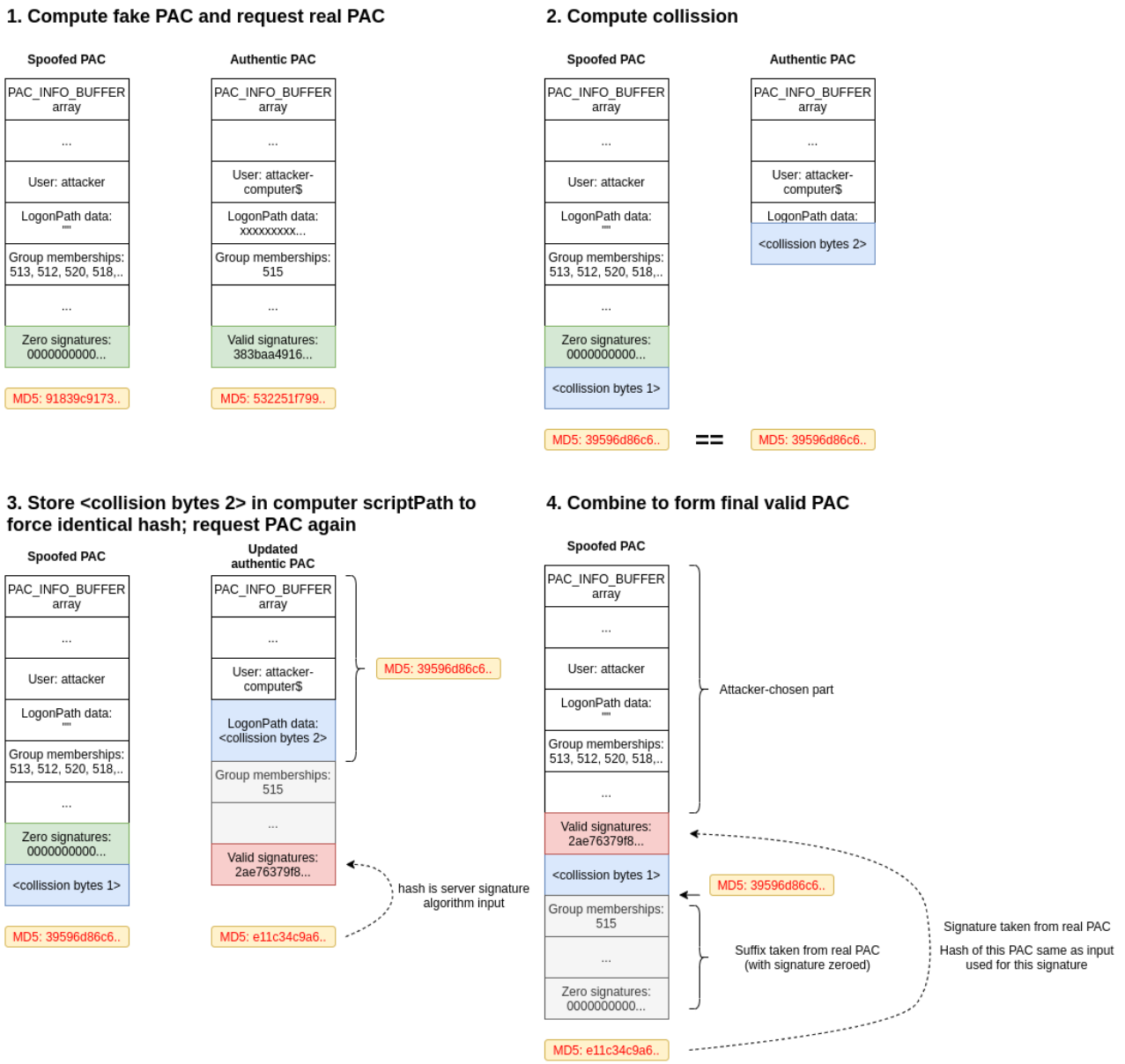

Figure 6: The four steps of the PAC spoofing attack.

A second vulnerability: spoofing with AES-HMAC-SHA1

The AES encryption types Windows support use a more secure CHKSUM operation: namely HMAC-SHA1 truncated to 96 bits (12 bytes). While SHA1 is also no longer collision-free, these operations apply the HMAC to a message directly and don't pre-hash it first. When an attacker does not know the key, they need around 2^{96} attempts before they can forge a MAC, which is probably enough in practice.

What's interesting about the RC4-HMAC exploit explained in the previous section is that is still works if the KDC signature is set using this HMAC-SHA1 scheme: since the KDC signature is only computed over the server signature (instead of the whole PAC) any collision in the server signature will result in a collision in the KDC signature, regardless of whether the MAC algorithm the latter uses is secure.

One could say that the security of the MAC-of-a-MAC scheme used for KDC signatures depends on the following three properties:

1. The KDC signature must be computed using a secure MAC algorithm.
2. The server signature must be computed using a secure MAC algorithm.
3. An attacker should not be able to produce two PAC's with the same server signature, **even if they know the MAC key used for the server signature**.

Property 1 holds when the KDC uses AES-HMAC-SHA1. Property 2 is broken when attackers can choose to use RC4-HMAC, but if an AES type were to be enforced this would be fixed.

However, property 3 does not necessarily follow from property 2: cryptographic MAC algorithms are designed to withstand forgery from attackers unaware of the key, but there's no standard requirement to prevent attackers who do know the key from producing specific messages with an identical MAC.

When HMAC output is not truncated and used with a collision-free hash function, it does happen to possess property 3. However, once you start chopping off bytes from the HMAC output (which may be fine in a typical scenario where the attacker does not know the key) you open the opportunity for *birthday attacks*.

Due to the birthday paradox, when an attacker produces two 96-bit HMAC's with some known key K and two random distinct messages there is an approximate chance of 1 in 2^{48} that the two HMAC's have the same value. This means that if an attacker can compute around 2^{48} pairs of HMAC's with a known key they should be able to find a collision and therefore spoof an AES-HMAC-SHA1 KDC signature. If someone were to write an optimized implementation, computing the +/- 2^{50} SHA-1 computations needed for this collision should be possible within two days on eight Nvidia GTX 1080 GPUs. This is significantly more time than the few hours needed to compute a chosen-prefix MD5 collision, but still very practical.

Impact

Abusing constrained delegation is already a well-known offensive technique. When you compromise an account A that has delegation privileges to some other account B, you will often be able to compromise B as well by impersonating a highly-privileged user. Only in quite hardened environments, where delegation is forbidden for important users, PAC spoofing may offer a useful advantage to the attacker.

In practice, I think it is pretty unlikely that this constrained delegation attack will be actually used by pentesters, red teamers or malicious attackers. The situations in which it offers an advantage are limited, the attack is difficult to perform and it requires spending a hours on computing an MD5 collision.

It should also be noted that the vulnerable RC4-HMAC cipher is used for transport encryption within various other protocols, like LDAP and DCE/RPC (which in turn is used as a transport layer for many protocols). Also, while RC4-HMAC was designed for use in Windows, it is also supported by alternative Kerberos implementations like MIT Kerberos and Heimdal, so protocols in non-Windows environments may be affected as well. While I have not yet been able to find a practical exploit in another protocol that uses RC4-HMAC, that does not mean that such exploits will not be discovered in the future.

The mitigations

These vulnerabilities were disclosed to Microsoft in May 2021. They were assigned CVE-2022-37966 and CVE-2022-37967 and patches were released on patch Tuesday of November 2022. One and a half year is a long time for a bug fix, but note that these are issues in protocols that a lot of systems depend on. Maintaining backwards compatibility and testing everything depending on these protocol must have been quite a pain! Despite this, the patches apparently still broke some things.

The first mitigation measure provided by the patch was to change the Kerberos encryption types accounts support for their session keys by default. This default now only includes AES types and not RC4-HMAC. Administrators can still choose to re-introduce default RC4-HMAC support by changing a new registry key named **DefaultDomainSupportedEncTypes**. Furthermore, these defaults will not apply to accounts that have a specific value set for **ms-DS-SupportedEncryptionType**, which may enable RC4-HMAC for specific accounts. Microsoft recommends searching your domain for these types of accounts and explicitly disabling the cipher on them.

A second mitigation, addressing the general PAC spoofing vulnerability, was to add an additional KDC signature to the PAC data structure. This signature would cover the entire PAC and not just the server signature.

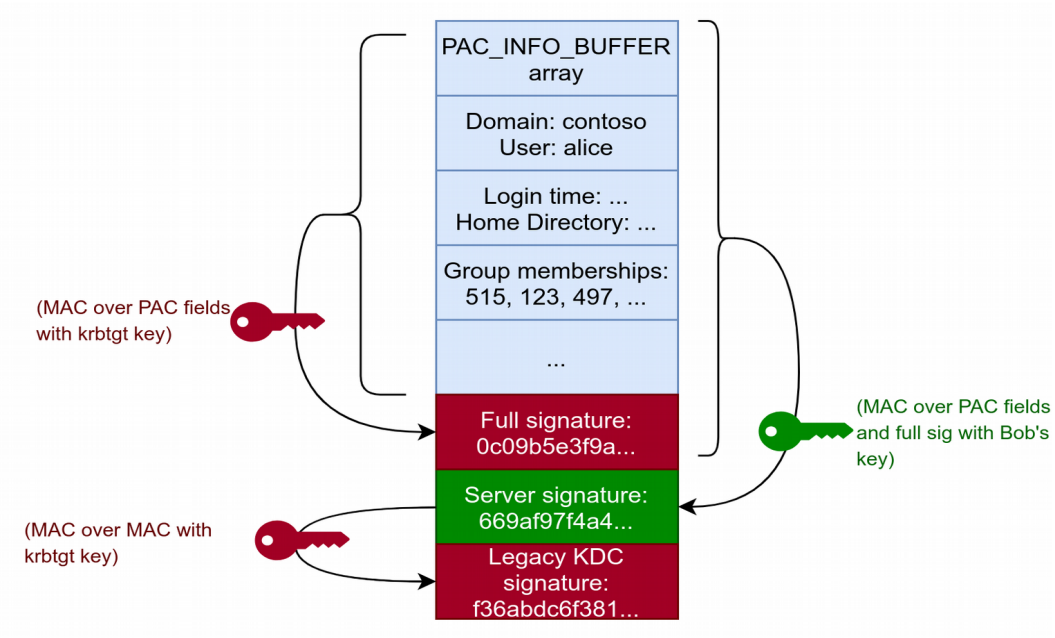

Figure 7: The additional KDC signature added to PAC's. This time covering the entire PAC contents are covered, so it is no longer necessary to assume attackers can not create colliding server signatures.

Be aware, however, that the November patch only causes the signature to be added, but does not enforce validation by default. How this signature is treated depends on the value of the **KrbtgtFullPacSignature** registry key. This key can be set to “audit mode” in order to receive alerts (event ID’s 43 and 44) whenever a PAC is received that either does not contain this signature or has a full signature that is invalid. If you want to actively block offending PAC’s (and completely mitigate the attack), this key should be set to “enforcement mode”.

Over the coming year, Microsoft will keep updating the default of this registry value until it will be set to enforcement mode in July 2023. Then, in October 2023, the possibility to reconfigure this

setting will be completely removed and full PAC signature validation is mandatory. Refer to Microsoft's advisory for the full timeline.

Be aware that as long as enforcement mode is not active your domain will **remain vulnerable to this attack**. If your domain uses constrained delegation but restricts it for sensitive users, you may want to consider switching to enforcement mode early.

While in audit mode, you may start receiving events related to older service tickets that had been issued before the November update. Once you start receiving these alerts after these tickets have been expired, this may be indicative of an attack.

Conclusion

I have identified cryptographic protocol vulnerabilities in the Kerberos encryption type RC4-HMAC, and the PAC Kerberos extension. I also discovered a practical exploit that, while having low impact, demonstrates that these vulnerabilities are not purely theoretical. Interestingly, this attack also shows that reliance on the MD5 hash function has not yet been fully eliminated from modern critical systems, and that hash collision attacks can be still be used as part of a practical exploit.

While RC4-HMAC has long been considered a ‘weaker’ encryption type than the newer AES-based alternatives (and has been deprecated by the IETF), I hope that this work provides a strong argument to eventually fully disable this encryption type in both Active Directory and other Kerberos environments, just like what happened to the encryption types based on DES. Microsoft’s choice to no longer enable the cipher for accounts by default is a good first step in this direction. As a nice side-effect, phasing out RC4 will also make common attack techniques such as Kerberoasting significantly more difficult.