CephFS - Feature #47162

mds: handle encrypted filenames in the MDS for fscrypt

08/26/2020 07:49 PM - Jeff Layton

Status:	Resolved	**% Done:**	0%
Priority:	Normal		
Assignee:	Xiubo Li		
Category:			
Target version:	v16.0.0		
Source:	Development	**Affected Versions:**	
Tags:		**Component(FS):**	MDS, libcephfs
Backport:		**Labels (FS):**	
Reviewed:		**Pull request ID:**	37297

Description

Once you turn a filename into encrypted text then it can have non-legal and non-printable embedded characters. To make that less painful, fscrypt converts the encrypted text to something legal and printable using a modified base64 encoding scheme. This means that the result may be longer than NAME_MAX (255 characters). To work around this, it base64 encodes the first ~150 bytes or so, hashes the rest of the name with sha256 and appends that to the first part:

https://www.kernel.org/doc/html/latest/filesystems/fscrypt.html#filename-hashing-and-encoding

The MDS will need to be able to deal with these special names in some fashion. I think there are two possible basic approaches (the first was my original thought, and then Zheng suggested the other):

1) The MDS can store the filenames as full-length base64 encoded strings. The MDS doesn't care about or enforce NAME_MAX, so this is possible, particularly if we bar legacy clients from accessing crypted directories altogether. MDS might need to be able to resolve names for clients that don't have a key though, so it may need to allow lookup by shortened/hashed name as well, and the client would need to tell the MDS which mode to use. If the MDS gets a hashed name, then it may have to do a slow-path lookup where it finds everything that matches the first part of the name, and then runs the hash itself on the rest to derive the name.

2) the MDS could always just use the shortened/hashed names. If we go this route then we'll need a way to store the full name of the file in the dentry so that we can satisfy readdir. MDS ops that create new dentries will need a way to pass both names to the MDS as well.

Note that when the cleartext names are "short" (<150 bytes or so) then both versions of the name are identical, So, we probably are ok to just have some special handling when the names are long.

I'm not particularly attached to either, but the second one is probably the simpler approach, providing adding a new dentry field for an alternate name is a sane thing to do. The two approaches are quite different though, and they have implications for the kclient so we probably need to settle on one or the other.

History

#1 - 08/31/2020 01:39 PM - Patrick Donnelly

- Subject changed from handle encrypted filenames in the MDS for fscrypt to mds: handle encrypted filenames in the MDS for fscrypt

- Assignee set to Xiubo Li

- Target version set to v16.0.0

- Source set to Development

#2 - 09/04/2020 01:15 PM - Xiubo Li

Will start it next week.

#3 - 09/09/2020 12:26 PM - Xiubo Li

- Status changed from New to In Progress

#4 - 09/16/2020 02:22 AM - Xiubo Li

From the source code, the encoded filename length will be roughly increased to 4/3 of the original filename.

```
/**
 * base64_encode() - base64-encode some bytes
 * @src: the bytes to encode
 * @len: number of bytes to encode
 * @dst: (output) the base64-encoded string.  Not NUL-terminated.
 *
 * Encodes the input string using characters from the set [A-Za-z0-9+,].

 * The encoded string is roughly 4/3 times the size of the input string.
 *
 * Return: length of the encoded string
 */
```

This will be filename presented from readdir() and it only take the ciphertext's first FSCRYPT_NOKEY_NAME_MAX bytes, which is offsetofend(struct fscrypt_nokey_name, sha256) == 157, to do the base64-encoding.

The ciphertext length will be:

```
struct fscrypt_nokey_name {
        u32 dirhash[2];
        u8 bytes[149];
        u8 sha256[SHA256_DIGEST_SIZE];
}; /* 189 bytes => 252 bytes base64-encoded, which is <= NAME_MAX (255) */
```

#5 - 09/16/2020 06:13 AM - Xiubo Li

Hi Jeff,

One question:

Currently the ext4 will just store the ciphertext as the final filename to the disk, and the max length of the ciphertext will be equal to or larger than the original filename length, but both will less than NAME_MAX.

Because the ciphertext may contains the '\0' and '/', which are not printable. So for ceph, from your kclient patches, you're planing to use the base64-encoded text, which will be directly encoded from the raw ciphertext instead without making sha256 for ciphertext[150]~ciphertext[end], right ?

And if the filename length is large enough then the base64-encoded plaintext maybe "ciphertext_length * 4 / 3" > 255 bytes ?

In another word, for ceph the final filename on disk will be user-copied-filename --> ciphertext --> based64-encoded-plaintext. But the vfs will do present or lookup the filename as "struct fscrypt_nokey_name" form. So for ceph we must store both the "based64-encoded-plaintext" and the "struct fscrypt_nokey_name" form one.

Why must we store the based64-encoded-plaintext is because that it will be used to dencrypt to the normal filename back, while the "struct fscrypt_nokey_name" form couldn't, because the [150, end] bytes has been coverted to sha256.

Is my understanding right ?

Thanks
BRs

#6 - 09/16/2020 08:10 AM - Xiubo Li

For the 2nd approach, suggeset by Zheng, more detail in my mind is:

If we will store both the "based64-encoded-plaintext" and shortened/hashed names.

Such as for "/mnt/cephfs/mydir/" and there has a "file0" under it and its based64-encoded name is "6Lbbm78gId17Blle0Uwz+a6I2QiBQR8Zzzc6QWBTYm,JkVzr,BqsSD". We will tell MDSs to store both the file0's ciphertext's base64-encoded value and "6Lbbm78gId17Blle0Uwz+a6I2QiBQR8Zzzc6QWBTYm,JkVzr,BqsSD".

For readdir():

No matter the key is present or not. The readdir("mydir/") will always return dentries' "base64-encoded-plaintext" full-name from the MDS side. And then if the key is present, the kclient will base64-decode and dencrypt it back to normal name "file0", or it will just base64-decode it back to ciphertext and then covert it to "fscrypt_nokey_name" and then based64-encodes it to "6Lbbm78gId17Blle0Uwz+a6I2QiBQR8Zzzc6QWBTYm,JkVzr,BqsSD".

For ->lookup():

In case the key is present, kclient will do encrypt "file0" --> covert it to "fscrypt_nokey_name" form --> base64-encodes it to "6Lbbm78gId17Blle0Uwz+a6I2QiBQR8Zzzc6QWBTYm,JkVzr,BqsSD", and then tell MDS to search "6Lbbm78gId17Blle0Uwz+a6I2QiBQR8Zzzc6QWBTYm,JkVzr,BqsSD" dentry in "/mydir/".

In case the key is absent, we can use the "6Lbbm78gId17Blle0Uwz+a6I2QiBQR8Zzzc6QWBTYm,JkVzr,BqsSD" directly.

With this apporach there is no extra work to do in MDS side if the names are long. And the maxlen of the shortened/hashed names will be 254. All the extra work for long names will be in kclient side.

With this approach there seems no need to covert the ciphertext to base64-encode text when storing it, we can just store the raw ciphertext in dentries ?

Thanks.

#7 - 09/16/2020 11:16 AM - Jeff Layton

Xiubo Li wrote:

> Hi Jeff,
>
> One question:
>
> Currently the ext4 will just store the ciphertext as the final filename to the disk, and the max length of the ciphertext will be equal to or larger than the original filename length, but both will less than NAME_MAX.
>
> Because the ciphertext may contains the '\0' and '/', which are not printable. So for ceph, from your kclient patches, you're planing to use the base64-encoded text, which will be directly encoded from the raw ciphertext instead without making sha256 for ciphertext[150]~ciphertext[end], right ?
>
> And if the filename length is large enough then the base64-encoded plaintext maybe "ciphertext_length * 4 / 3" > 255 bytes ?
>
> In another word, for ceph the final filename on disk will be user-copied-filename --> ciphertext --> based64-encoded-plaintext. But the vfs will do present or lookup the filename as "struct fscrypt_nokey_name" form. So for ceph we must store both the "based64-encoded-plaintext" and the "struct fscrypt_nokey_name" form one.
>
> Why must we store the based64-encoded-plaintext is because that it will be used to dencrypt to the normal filename back, while the "struct fscrypt_nokey_name" form couldn't, because the [150, end] bytes has been coverted to sha256.
>
> Is my understanding right ?

Yes, basically.

In principle, if the name is stored using base64-encoded ciphertext names, then we can always derive the "nokey name" (with the hash at the end). So, we don't *technically* need to store both versions of those names, but it may be simpler to do so.

#8 - 09/16/2020 11:25 AM - Jeff Layton

Xiubo Li wrote:

> With this approach there seems no need to covert the ciphertext to base64-encode text when storing it, we can just store the raw ciphertext in dentries ?

Correct.

If we're adding a new field to hold the full name, then we can just have it store the binary data and a length. That's going to be more space efficient on the MDS as well.

For names that are short, we can probably also skip storing the full (binary) name altogether. Names that are that long are pretty rare, and that should mean that we only need to send and store two versions of the name in rare cases.

#9 - 09/16/2020 12:27 PM - Jeff Layton

Xiubo Li wrote:

```
struct fscrypt_nokey_name {
        u32 dirhash[2];
        u8 bytes[149];
        u8 sha256[SHA256_DIGEST_SIZE];
}; /* 189 bytes => 252 bytes base64-encoded, which is <= NAME_MAX (255) */
```

One thing to note here too is that we have an extra 64 bits that can be used to store identifying information in the dirhash field. Most local fs' use that to store a dirhash that can be used to look up the correct dentry.

I didn't see a way to use that in ceph, but it is available if you do.

#10 - 09/17/2020 05:09 AM - Xiubo Li

Jeff Layton wrote:

> Xiubo Li wrote:
>
>> Hi Jeff,
>>
>> One question:
>>
>> Currently the ext4 will just store the ciphertext as the final filename to the disk, and the max length of the ciphertext will be equal to or larger than the original filename length, but both will less than NAME_MAX.
>>
>> Because the ciphertext may contains the '\0' and '/', which are not printable. So for ceph, from your kclient patches, you're planing to use the base64-encoded text, which will be directly encoded from the raw ciphertext instead without making sha256 for ciphertext[150] ~ciphertext[end], right ?
>>
>> And if the filename length is large enough then the base64-encoded plaintext maybe "ciphertext_length * 4 / 3" > 255 bytes ?
>>
>> In another word, for ceph the final filename on disk will be user-copied-filename --> ciphertext --> based64-encoded-plaintext. But the vfs will do present or lookup the filename as "struct fscrypt_nokey_name" form. So for ceph we must store both the "based64-encoded-plaintext" and the "struct fscrypt_nokey_name" form one.
>>
>> Why must we store the based64-encoded-plaintext is because that it will be used to dencrypt to the normal filename back, while the "struct fscrypt_nokey_name" form couldn't, because the [150, end] bytes has been coverted to sha256.
>>
>> Is my understanding right ?

> Yes, basically.
>
> In principle, if the name is stored using base64-encoded ciphertext names, then we can always derive the "nokey name" (with the hash at the end). So, we don't *technically* need to store both versions of those names, but it may be simpler to do so.

Yeah, right.

If the master key is absent, for the ->lookup() the client will tell MDS the "nokey name", and in MDS side it must compute the "nokey name" for all the searched dentries, right ?

Then for the improved approach, let's assume the approach(A) will be:

1), We will store the binary ciphertext names in dentries only and the kclient will just tell MDS the ciphertext and the MDS will compute the base64-encoded "nokey name" and cache them in memories only. And the base64-encoded "nokey name" could be used for in debug logs and quick ->lookup().

2), When fetching the dentries, the MDS will compute the base64-encoded "nokey name" for each dentry and cache them in memories, and then they could be used for quick ->lookup() and debug logs.

For the 1), maybe we can compute the base64-encoded "nokey name" in kclient side to reduce the MDS's overload.

And also IMO we should add one flag in the dentries to tell the MDS what's the version of the "nokey name" it is using ? As Eric Biggers mentioned in your patches if the "nokey name" format could be changed in future, we need to compatiable with old version.

And for the first 64 bits of the "nokey name", we can use them as version flags, what version it is when kclient sending the "nokey name" to MDS. Currently we can just leave them as 0 for the cuerrent version.

An alternative approach(B) will be:

1), We will store both the binary ciphertext names and the base64-encoded "nokey name" in dentries.
2), When fetching the dentries, there is no need to do compute the base64-encoded "nokey name" in MDS.

For this approache, there is no extra version flag need in dentry, the base64-encoded "nokey name" has already contain it.

For the above 2 approaches, there is no special work for the long name case.

> If we're adding a new field to hold the full name, then we can just have it store the binary data and a length. That's going to be more space efficient on the MDS as well.

> For names that are short, we can probably also skip storing the full (binary) name altogether. Names that are that long are pretty rare, and that should mean that we only need to send and store two versions of the name in rare cases.

Yeah, since the long name case is rare, and for short name case, we can derive the normal file names from both binary ciphertext and base64-encoded "nokey name", so for the 3rd approach(C) we could do:

1), For the short name case, we will only store the base64-encoded "nokey name".

2), For the rare long name case, we will store both the base64-encoded "nokey name" and ciphertext binary.

For this apporach, if the kclient using a higher version, we need to covert "nokey name" version to local one, and then do the match compare.

But we may switch the fs/scrypt "nokey name" to ceph's private "nokey name" format, which won't contain the dirhash[2], since it useless for ceph. Then for the above approaches there is no need to care about the versions.

For the above 3 approaches, which one should we use ? Then I will begin to work on this in MDS side.

Thanks.

#11 - 09/17/2020 05:19 AM - Xiubo Li

Xiubo Li wrote:

> Jeff Layton wrote:
>
> > Xiubo Li wrote:

[...]

> Yeah, since the long name case is rare, and for short name case, we can derive the normal file names from both binary ciphertext and base64-encoded "nokey name", so for the 3rd approach(C) we could do:
>
> 1), For the short name case, we will only store the base64-encoded "nokey name".
>
> 2), For the rare long name case, we will store both the base64-encoded "nokey name" and ciphertext binary.

For the approach(C) for the rare long name case, we can also just store the base64-encoded "nokey name" and ciphertext[150 ~ end] binary, no need the full cephertext binary data.

#12 - 09/17/2020 03:36 PM - Jeff Layton

Xiubo Li wrote:

> Yeah, right.
>
> If the master key is absent, for the ->lookup() the client will tell MDS the "nokey name", and in MDS side it must compute the "nokey name" for all the searched dentries, right ?

Yes, unless it's already been stored somewhere.

> Then for the improved approach, let's assume the approach(A) will be:
>
> 1), We will store the binary ciphertext names in dentries only and the kclient will just tell MDS the ciphertext and the MDS will compute the base64-encoded "nokey name" and cache them in memories only. And the base64-encoded "nokey name" could be used for in debug logs and quick ->lookup().
>
> 2), When fetching the dentries, the MDS will compute the base64-encoded "nokey name" for each dentry and cache them in memories, and then they could be used for quick ->lookup() and debug logs.
>
> For the 1), maybe we can compute the base64-encoded "nokey name" in kclient side to reduce the MDS's overload.
>
> And also IMO we should add one flag in the dentries to tell the MDS what's the version of the "nokey name" it is using ? As Eric Biggers mentioned in your patches if the "nokey name" format could be changed in future, we need to compatiable with old version.

> And for the first 64 bits of the "nokey name", we can use them as version flags, what version it is when kclient sending the "nokey name" to MDS. Currently we can just leave them as 0 for the cuerrent version.

Good idea. I think we'll want a version field instead of a flag (as you mention below).

Note too that we don't necessarily need to use the nokey_name format to store them. We can modify it for our purposes as long as we can convert from the current nokey name to the one that ceph will use. For instance, the dirhash field is not terribly useful for ceph, so there's no need to send that across the wire or store it.

> An alternative approach(B) will be:
>
> 1), We will store both the binary ciphertext names and the base64-encoded "nokey name" in dentries.
> 2), When fetching the dentries, there is no need to do compute the base64-encoded "nokey name" in MDS.
>
> For this approache, there is no extra version flag need in dentry, the base64-encoded "nokey name" has already contain it.
>
> For the above 2 approaches, there is no special work for the long name case.
>
> > If we're adding a new field to hold the full name, then we can just have it store the binary data and a length. That's going to be more space efficient on the MDS as well.
> >
> > For names that are short, we can probably also skip storing the full (binary) name altogether. Names that are that long are pretty rare, and that should mean that we only need to send and store two versions of the name in rare cases.
>
> Yeah, since the long name case is rare, and for short name case, we can derive the normal file names from both binary ciphertext and base64-encoded "nokey name", so for the 3rd approach(C) we could do:
>
> 1), For the short name case, we will only store the base64-encoded "nokey name".
>
> 2), For the rare long name case, we will store both the base64-encoded "nokey name" and ciphertext binary.
>
> For this apporach, if the kclient using a higher version, we need to covert "nokey name" version to local one, and then do the match compare.
>
> But we may switch the fs/scrypt "nokey name" to ceph's private "nokey name" format, which won't contain the dirhash[2], since it useless for ceph. Then for the above approaches there is no need to care about the versions.
>
> For the above 3 approaches, which one should we use ? Then I will begin to work on this in MDS side.

Probably something like the last one. I think we're best off avoiding any logic that requires the MDS to stop treating dentry names as opaque for lookups.

Here's what I'd suggest:

Add an "alternate name" field to the dentry stored by the MDS, and rev the MClientRequest object version to allow dentries to be created and reported with that alternate field as well. Think of it like an optional xattr that's attached to a dentry instead of an inode. The MDS won't need to look up names by that, but will report them in readdir and in traces.

The client is then responsible for creating and fetching dentries with our ceph variant of the nokey name format, but will pass along the alternate name when creating a dentry, when the name is long and it's required.

When doing a readdir, the MDS will need to send these fields along in the response (if a dentry has one), so those responses may also need a new version. The client can then decrypt those when it has a key and create dentries like it should. Otherwise it will just base64 decode and decrypt the regular dentry name if the alternate name isn't present.

I think that approach will give us the most flexibility going forward. We can change the hashing or whatever, and the MDS shouldn't need to do anything special as long as the clients all support the new format.

Maybe we can bump the MClientRequest HEAD_VERSION with a path3 field? When that's set on an operation that creates a dentry, the MDS can attach the blob to the resulting dentry. We'll also need to look at how to extend readdir responses with this alternate_name field and in dentries in traces.

#13 - 09/17/2020 03:52 PM - Zheng Yan

Jeff Layton wrote:

> Probably something like the last one. I think we're best off avoiding any logic that requires the MDS to stop treating dentry names as opaque for lookups.
>
> Here's what I'd suggest:
>
> Add an "alternate name" field to the dentry stored by the MDS, and rev the MClientRequest object version to allow dentries to be created and reported with that alternate field as well. Think of it like an optional xattr that's attached to a dentry instead of an inode. The MDS won't need to look up names by that, but will report them in readdir and in traces.
>
> The client is then responsible for creating and fetching creating dentries with our ceph variant of the nokey name format, but will pass along the alternate name when creating a dentry, when the name is long and it's required.
>
> When doing a readdir, the MDS will need to send these fields along in the response (if a dentry has one), so those responses may also need a new version. The client can then decrypt those when it has a key and create dentries like it should. Otherwise it will just base64 decode and decrypt the regular dentry name if the alternate name isn't present.
>
> I think that approach will give us the most flexibility going forward. We can change the hashing or whatever, and the MDS shouldn't need to do anything special as long as the clients all support the new format.
>
> Maybe we can bump the MClientRequest HEAD_VERSION with a path3 field? When that's set on an operation that creates a dentry, the MDS can attach the blob to the resulting dentry. We'll also need to look at how to extend readdir responses with this alternate_name field and in dentries in traces.

For adding the blob in request, we can encode it to req->r_pagelist (after normal xattrs). For attaching the blob in reply, we can extend dentry lease.

#14 - 09/18/2020 10:39 AM - Xiubo Li

Jeff Layton wrote:

> Xiubo Li wrote:
>
> > Yeah, right.
> >
> > If the master key is absent, for the ->lookup() the client will tell MDS the "nokey name", and in MDS side it must compute the "nokey name" for all the searched dentries, right ?
>
> Yes, unless it's already been stored somewhere.

Yeah.

[...]

> And for the first 64 bits of the "nokey name", we can use them as version flags, what version it is when kclient sending the "nokey name" to MDS. Currently we can just leave them as 0 for the cuerrent version.

> Good idea. I think we'll want a version field instead of a flag (as you mention below).
>
> Note too that we don't necessarily need to use the nokey_name format to store them. We can modify it for our purposes as long as we can convert from the current nokey name to the one that ceph will use. For instance, the dirhash field is not terribly useful for ceph, so there's no need to send that across the wire or store it.

Yeah, agree.

In kclient, the ceph nokey name could be:

```
struct ceph_fscrypt_nokey_name {
    u8          version; // will be 1 for the first version

    // keep the following 2 feilds the same with fs/fscrypt's nokey name
    u8          bytes[149];
    u8          sha254[32];
};
```

I kclinet side, for both short/long name cases, just base64-encoded above nokey name and save it in the MClientRequest's "r_path1".

Yeah, as Zheng menthioned, we could use the "req->r_pagelist" to send the "alternate_name", no need to bump the HEAD_VERSION of MClientRequest ?

We can reuse the "r_path2" and set it to "ciphertext" or "alternate_name_ciphertext", and append the ciphertext bianry in r_pagelist. Because when creating a dentry, there should no attribute will be set, so the r_path2 will be useless, right ?

[...]

> But we may switch the fs/scrypt "nokey name" to ceph's private "nokey name" format, which won't contain the dirhash[2], since it useless for ceph. Then for the above approaches there is no need to care about the versions.
>
> For the above 3 approaches, which one should we use ? Then I will begin to work on this in MDS side.

> Probably something like the last one. I think we're best off avoiding any logic that requires the MDS to stop treating dentry names as opaque for lookups.

Agree.

> Here's what I'd suggest:
>
> Add an "alternate name" field to the dentry stored by the MDS, and rev the MClientRequest object version to allow dentries to be created and reported with that alternate field as well. Think of it like an optional xattr that's attached to a dentry instead of an inode. The MDS won't need to look up names by that, but will report them in readdir and in traces.

> The client is then responsible for creating and fetching dentries with our ceph variant of the nokey name format, but will pass along the alternate name when creating a dentry, when the name is long and it's required.

Yeah, this looks good.

BTW, what the alternat_name will store ? The full ciphertext binary or the full base64-encoded ciphertext ?

I assume it should be The full raw ciphertext binary, since only the kclient will use it, so no need to base64-encode it.

> When doing a readdir, the MDS will need to send these fields along in the response (if a dentry has one), so those responses may also need a new version. The client can then decrypt those when it has a key and create dentries like it should. Otherwise it will just base64 decode and decrypt the regular dentry name if the alternate name isn't present.

For the responses, maybe we can just do:
1), for short name case, always send the "ceph_fscrypt_nokey_name" to kclient.
2), for long name case, maybe we could send the "alternate_name" ? Because we can derive anything needed from it. With this there is no need to check the nokey name version. Or just send both, because the long name case is rare, and it won't cost to much network overload ?

> I think that approach will give us the most flexibility going forward. We can change the hashing or whatever, and the MDS shouldn't need to do anything special as long as the clients all support the new format.
>
> Maybe we can bump the MClientRequest HEAD_VERSION with a path3 field? When that's set on an operation that creates a dentry, the MDS can attach the blob to the resulting dentry. We'll also need to look at how to extend readdir responses with this alternate_name field and in dentries in traces.

#15 - 09/18/2020 11:04 AM - Xiubo Li

Zheng Yan wrote:

> Jeff Layton wrote:

[...]

> > I think that approach will give us the most flexibility going forward. We can change the hashing or whatever, and the MDS shouldn't need to do anything special as long as the clients all support the new format.
> >
> > Maybe we can bump the MClientRequest HEAD_VERSION with a path3 field? When that's set on an operation that creates a dentry, the MDS can attach the blob to the resulting dentry. We'll also need to look at how to extend readdir responses with this alternate_name field and in dentries in traces.
>
> For adding the blob in request, we can encode it to req->r_pagelist (after normal xattrs). For attaching the blob in reply, we can extend dentry lease.

Hi Zheng, Jeff,

For the reply one, maybe to add a new field will be better ?

#16 - 09/18/2020 11:55 AM - Jeff Layton

Xiubo Li wrote:

> Yeah, this looks good.
>
> BTW, what the alternat_name will store ? The full ciphertext binary or the full base64-encoded ciphertext ?
>
> I assume it should be The full raw ciphertext binary, since only the kclient will use it, so no need to base64-encode it.

Yes. I don't see a need to base64 encode this field since legacy clients should just ignore it.

> For the responses, maybe we can just do:
> 1), for short name case, always send the "ceph_fscrypt_nokey_name" to kclient.
> 2), for long name case, maybe we could send the "alternate_name" ? Because we can derive anything needed from it. With this there is no need to check the nokey name version. Or just send both, because the long name case is rare, and it won't cost to much network overload ?

I'd send both versions for long names. If you don't, then you'll need to teach the MDS how to convert from full ciphertext name to ceph_fscrypt_nokey_name and at that point it's not treating it as opaque anymore.

I think just treating this as an (opaque) alternate name field is the simplest way to do this. The MDS should never even need to look at this field -- only send it back to the client in readdir if it exists.

#17 - 09/20/2020 11:01 AM - Xiubo Li

Jeff Layton wrote:

> Xiubo Li wrote:
>
>> Yeah, this looks good.
>>
>> BTW, what the alternat_name will store ? The full ciphertext binary or the full base64-encoded ciphertext ?
>>
>> I assume it should be The full raw ciphertext binary, since only the kclient will use it, so no need to base64-encode it.
>
> Yes. I don't see a need to base64 encode this field since legacy clients should just ignore it.

Cool.

>> For the responses, maybe we can just do:
>> 1), for short name case, always send the "ceph_fscrypt_nokey_name" to kclient.
>> 2), for long name case, maybe we could send the "alternate_name" ? Because we can derive anything needed from it. With this there is no need to check the nokey name version. Or just send both, because the long name case is rare, and it won't cost to much network overload ?
>
> I'd send both versions for long names. If you don't, then you'll need to teach the MDS how to convert from full ciphertext name to ceph_fscrypt_nokey_name and at that point it's not treating it as opaque anymore.

Yeah, agree. Or we need to copy some kernel code to MDS.

> I think just treating this as an (opaque) alternate name field is the simplest way to do this. The MDS should never even need to look at this field -- only send it back to the client in readdir if it exists.

Okay.

#18 - 09/21/2020 04:07 AM - Xiubo Li

Ceph has its own base64 encode/decode logic already in src/common/armor.c, which is the same with the kernel does.

#19 - 09/21/2020 01:39 PM - Xiubo Li

I am planing to append a `fscrypt.alternate_name : ${raw_ciphertext}` pair to the xattr map when doing the create dentries in long name case. And for readdir(), also just append the above pair in the xattr map for each dentry in the MClientReply.

Both the MDS and client will just ignore this fake xattr pair. And also will add one CEPHFS_FEATURE_FSCRYPTED bit to make the client to compatible with old MDS daemons.

#20 - 09/21/2020 01:42 PM - Jeff Layton

Xiubo Li wrote:

> Ceph has its own base64 encode/decode logic already in src/common/armor.c, which is the same with the kernel does.

Careful here. Most base64 implementations will generate encodings with '/' characters which are not legal in filenames (and may give you other troublesome chars).

I don't think we'll need to do any decoding if we just store the encrypted "alternate" name as raw binary data.

#21 - 09/21/2020 01:51 PM - Xiubo Li

Jeff Layton wrote:

> Xiubo Li wrote:
>
>> Ceph has its own base64 encode/decode logic already in src/common/armor.c, which is the same with the kernel does.
>
> Careful here. Most base64 implementations will generate encodings with '/' characters which are not legal in filenames (and may give you other troublesome chars).
>
> I don't think we'll need to do any decoding if we just store the encrypted "alternate" name as raw binary data.

Yeah, right. For the encryped ciphertext filenames, there is no need to do any encoding/decoding in MDS side.

Just find this in ceph and commented it here if we need it later :-)

#22 - 09/22/2020 02:35 AM - Xiubo Li

Hi Jeff,

There is another case for lookup:

If the MDS is old version, such as all the dentries is under `ceph_fscrypt_nokey_name` version 1, the current version.

```
struct ceph_fscrypt_nokey_name {
    u8          version; // will be 1 for the first version

    // keep the following 2 feilds the same with fs/fscrypt's nokey name
    u8          bytes[149];
    u8          sha254[32];
};
```

And if in the future the kclient is using the verion 2, it will be like:

```
struct ceph_fscrypt_nokey_name {
    u8          version; // 2

    // keep the following 2 feilds the same with fs/fscrypt's nokey name
    u8          bytes[141];
    u8          sha254[32];
};
```

The `bytes[N]` has changed to 141, the extra 8 bytes maybe used for the 128 bits dirhash[] by fs/scrypt/.

So in this case, should we do the versions comparation first in the MDS side when doing the lookup ?

Or should we always use the fixed `bytes[]` length in ceph in future versions ?

#23 - 09/22/2020 11:52 AM - Jeff Layton

I think the MDS should treat these names as opaque. The client should never need to look up a dentry by the binary crypttext name, and the MDS has no reason to ever need to look at this name itself.

It's just an extra blob of binary data that's attached to the dentry. The storage and interpretation of it should be left entirely to the client.

#24 - 09/22/2020 12:02 PM - Jeff Layton

Xiubo Li wrote:

> Hi Jeff,
>
> There is another case for lookup:
>
> If the MDS is old version, such as all the dentries is under `ceph_fscrypt_nokey_name` version 1, the current version.
>
> [...]
>
> And if in the future the kclient is using the verion 2, it will be like:
>
> [...]
>
> The `bytes[N]` has changed to 141, the extra 8 bytes maybe used for the 128 bits dirhash[] by fs/scrypt/.
>
> So in this case, should we do the versions comparation first in the MDS side when doing the lookup ?
>
> Or should we always use the fixed `bytes[]` length in ceph in future versions ?

Hmm ok, so this is for the "actual" dentry name, not the alternate name.

I'd say that we just have the MDS treat these names as opaque too. If we have a filesystem with v1 names in it, and it gets mounted by a kernel that understands v2 names, then the client would have to operate in "legacy mode" an use v1 names, or maybe reject the mount (if v1 support were deprecated).

If a client that's only capable of dealing with v1 names mounts a filesystem with v2 names, then we should probably reject the mount, or just not allow the key to be set somehow.

The tricky part here is that we don't have a way to set crypto metadata on the filesystem at large. It's possible though that we might be able to infer which version to use from the enryption.ctx xattr of the root inode, though for that we may need to extend fscrypt_context with a new version that has a field for that (or maybe just claim some of the __reserved fields in the v2 one).

#25 - 09/22/2020 02:10 PM - Xiubo Li

Jeff Layton wrote:

> Xiubo Li wrote:
>
> > Hi Jeff,
> >
> > There is another case for lookup:
> >
> > If the MDS is old version, such as all the dentries is under `ceph_fscrypt_nokey_name` version 1, the current version.
> >
> > [...]
> >
> > And if in the future the kclient is using the verion 2, it will be like:
> >
> > [...]
> >
> > The `bytes[N]` has changed to 141, the extra 8 bytes maybe used for the 128 bits dirhash[] by fs/scrypt/.
> >
> > So in this case, should we do the versions comparation first in the MDS side when doing the lookup ?
> >
> > Or should we always use the fixed `bytes[]` length in ceph in future versions ?
>
> Hmm ok, so this is for the "actual" dentry name, not the alternate name.

Yeah, right.

> I'd say that we just have the MDS treat these names as opaque too. If we have a filesystem with v1 names in it, and it gets mounted by a kernel that understands v2 names, then the client would have to operate in "legacy mode" an use v1 names, or maybe reject the mount (if v1 support were deprecated).
>
> If a client that's only capable of dealing with v1 names mounts a filesystem with v2 names, then we should probably reject the mount, or just not allow the key to be set somehow.

Okay, let's make it simple and just reject it if the versions do not match.

> The tricky part here is that we don't have a way to set crypto metadata on the filesystem at large. It's possible though that we might be able to infer which version to use from the enryption.ctx xattr of the root inode, though for that we may need to extend fscrypt_context with a new version that has a field for that (or maybe just claim some of the __reserved fields in the v2 one).

There has another case that, if there has two dirs, which are mounted in 2 different kclients, which maybe using different fscrypt versions, so I assume the "root" inode here you mentioned is the mount dir inodes ?

With this for the "struct ceph_fscrypt_nokey_name {}', the 'version' member is no need any more, right ?

```
struct ceph_fscrypt_nokey_name {
        // keep the following 2 feilds the same with fs/fscrypt's nokey name
    u8          bytes[149];
    u8          sha254[32];
};
```

#26 - 09/23/2020 01:24 AM - Xiubo Li

- Status changed from In Progress to Fix Under Review

- Pull request ID set to 37297

#27 - 09/23/2020 11:02 AM - Xiubo Li

Hi Jeff,

Have finished code in MDS, and for now I didn't handle the loopup version case. All the version related checking should be in kclient side.

#28 - 01/16/2021 09:11 PM - Patrick Donnelly

- Status changed from Fix Under Review to Resolved