rgw - Bug #21003

chunk signature mismatch for AWSv4 and Oracle Secure Backup Cloud Module

08/15/2017 01:01 PM - Dan van der Ster

Status:	In Progress	**Start date:**	08/15/2017
Priority:	Normal	**Due date:**	
Assignee:	Marcus Watts	**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:		**Spent time:**	0.00 hour
Source:	Community (user)	**Reviewed:**	
Tags:		**Affected Versions:**	
Backport:	luminous	**ceph-qa-suite:**	
Regression:	No	**Pull request ID:**	
Severity:	3 - minor		

Description

We are getting

```
AWSv4ComplMulti: ERROR: chunk signature mismatch
```

when trying to upload to 12.1.3 luminous rgw from Oracle Secure Backup Cloud Module for Amazon S3.

```
2017-08-15 11:42:25.523344 7fc6f5ba3700 20 AWSv4ComplMulti: ERROR: chunk signature mismatch
2017-08-15 11:42:25.523344 7fc6f5ba3700 20 AWSv4ComplMulti: declared signature=6617277a2d0b07638df
f7dfa7685d796a24187577bf6cbcdbafc91e396973749
2017-08-15 11:42:25.523345 7fc6f5ba3700 20 AWSv4ComplMulti: calculated signature=f0fdd6e8b2b3a7f9c
dc2cb27a7fe241373a4ca078399c993db2a1dce507dfe6b
```

The client log shows:

```
nhp:  > Authorization: <hidden>
nhp:  >
nhp:  > write 16384 bytes
nhpAWS4EncWrite: string-to-sign
AWS4-HMAC-SHA256-PAYLOAD
20170815T094225Z
20170815/us-east-1/s3/aws4_request
b0214c8625d6f2fc23306b43dfc88d7cbdab552e3ad01fb3de36acc540e9e31f
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
d34f8a88085f31d696649c059b85d6cc0ae84d05409ba6387e278f3642038d6d
nhp: 2017-08-15 11:42:25.520675 send enter 0x7f284d006740 2697
```

And the rgw log shows:

```
2017-08-15 11:42:25.522482 7fc6f5ba3700  2 req 46:0.001111:s3:PUT /file_chunk/0/SBTDB/unknown/2017
-08-15/test.txt/ZyH3tH
16jobK/0000000001:put_obj:executing
2017-08-15 11:42:25.522521 7fc6f5ba3700 20 parsed new chunk; signature=6617277a2d0b07638dff7dfa768
5d796a24187577bf6cbcdb
afc91e396973749, data_length=87, data_starts_in_stream=91
2017-08-15 11:42:25.522528 7fc6f5ba3700 20 AWSv4ComplMulti: filled=77
2017-08-15 11:42:25.523276 7fc6f5ba3700 20 AWSv4ComplMulti: filled=10
```

```
2017-08-15 11:42:25.523318 7fc6f5ba3700 20 AWSv4ComplMulti: string_to_sign=
AWS4-HMAC-SHA256-PAYLOAD
20170815T094225Z
20170815/us-east-1/s3/aws4_request
b0214c8625d6f2fc23306b43dfc88d7cbdab552e3ad01fb3de36acc540e9e31f
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
535be80def9bba552ec426b35731c8efa5d2b6c3aff76e87e3289747de136b65
2017-08-15 11:42:25.523344 7fc6f5ba3700 20 AWSv4ComplMulti: ERROR: chunk signature mismatch
2017-08-15 11:42:25.523344 7fc6f5ba3700 20 AWSv4ComplMulti: declared signature=6617277a2d0b07638df
f7dfa7685d796a24187577bf6cbcdbafc91e396973749
2017-08-15 11:42:25.523345 7fc6f5ba3700 20 AWSv4ComplMulti: calculated signature=f0fdd6e8b2b3a7f9c
dc2cb27a7fe241373a4ca078399c993db2a1dce507dfe6b
2017-08-15 11:42:25.523555 7fc6f5ba3700  2 req 46:0.002183:s3:PUT /file_chunk/0/SBTDB/unknown/2017
-08-15/test.txt/ZyH3tH16jobK/0000000001:put_obj:completing
2017-08-15 11:42:25.523602 7fc6f5ba3700  2 req 46:0.002230:s3:PUT /file_chunk/0/SBTDB/unknown/2017
-08-15/test.txt/ZyH3tH16jobK/0000000001:put_obj:op status=-2027
2017-08-15 11:42:25.523607 7fc6f5ba3700  2 req 46:0.002236:s3:PUT /file_chunk/0/SBTDB/unknown/2017
-08-15/test.txt/ZyH3tH16jobK/0000000001:put_obj:http status=403
```

Note that string_to_sign's differ by their payload_hash.

Related issues:

Related to rgw - Bug #21015: rgw: make HTTP dechunking compatible with Amazon S3	**Resolved**	**08/16/2017**

History

#1 - 08/15/2017 07:41 PM - Marcus Watts

- Assignee set to Marcus Watts

I think I've found a similar problem around object size = 16. I think it's likely to have the same cause. Object size 15 worked fine for me, object size 17 "worked" - but processed things one byte at a time. I'll put more here once I learn more what's happening.

#2 - 08/16/2017 02:24 AM - Marcus Watts

I have a possible fix for this in
https://github.com/ceph/ceph/pull/17040

The logic to compute "to_extract" was using the wrong "stream_pos", so it got confused depending on the chunk size.

#3 - 08/16/2017 12:52 PM - Dan van der Ster

- File wireshark.txt added

- File sbtio_28681_140566666970880.log added

- File ceph-client.rgw.cs3test.log added

Unfortunately this PR doesn't solve the problem for us. We attached the entire client and server log (running with your patch), and the wireshark dump from the client side.

The client starting the chunked upload and first two chunks:

```
STREAMING-AWS4-HMAC-SHA256-PAYLOAD
nhpAWS4Authentication: string-to-sign
AWS4-HMAC-SHA256
20170816T115603Z
20170816/us-east-1/s3/aws4_request
6a93f48bcc8d274117f48a50dc84c5cfd812b3fb7edc5be7d763459647423c62
nhp:  > Authorization: <hidden>
nhp:  >
nhp:  > write 16384 bytes
nhpAWS4EncWrite: string-to-sign
```

```
AWS4-HMAC-SHA256-PAYLOAD
20170816T115603Z
20170816/us-east-1/s3/aws4_request
f69993822618a847e0e101d9fff756471f3faed67808890e1b6fb536c80c98c0
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
b3d650f1ee6a4f131076f307ac5eb6ff9563ab3714eacf963e093238733990d4
nhp: 2017-08-16 13:56:03.569239 send enter 0x7fd8353ec740 2677
nhp: 2017-08-16 13:56:03.569292 send return 2677 0 0 elapsed +0 00:00:00.000053
nhp: 2017-08-16 13:56:03.569300 send enter 0x7fd8353ec740 4
nhp: 2017-08-16 13:56:03.569310 send return 4 0 0 elapsed +0 00:00:00.000010
nhp: 2017-08-16 13:56:03.569315 send enter 0x7fd8353ec740 87
nhp: 2017-08-16 13:56:03.569328 send return 87 0 0 elapsed +0 00:00:00.000013
nhp: 2017-08-16 13:56:03.569333 send enter 0x7fd8353ec740 2
nhp: 2017-08-16 13:56:03.569343 send return 2 0 0 elapsed +0 00:00:00.000010
nhp: 2017-08-16 13:56:03.569349 send enter 0x7fd8353ec740 6
nhp: 2017-08-16 13:56:03.569357 send return 6 0 0 elapsed +0 00:00:00.000008
nhp: 2017-08-16 13:56:03.569362 send enter 0x7fd8353ec740 16384
nhp: 2017-08-16 13:56:03.569381 send return 16384 0 0 elapsed +0 00:00:00.000019
nhp: 2017-08-16 13:56:03.569387 send enter 0x7fd8353ec740 2
nhp: 2017-08-16 13:56:03.569393 send return 2 0 0 elapsed +0 00:00:00.000006
nhp:  > write 16384 bytes
nhpAWS4EncWrite: string-to-sign
AWS4-HMAC-SHA256-PAYLOAD
20170816T115603Z
20170816/us-east-1/s3/aws4_request
d61f8acc559fdf781581b6eda77de720360a2786f1d8449aa10654cf73997487
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
e33f24140499430b048f6600af4f41f3ccb0cb766d9f7661124cf8ba4b827523
nhp: 2017-08-16 13:56:03.569633 send enter 0x7fd8353ec740 4
nhp: 2017-08-16 13:56:03.569654 send return 4 0 0 elapsed +0 00:00:00.000021
nhp: 2017-08-16 13:56:03.569659 send enter 0x7fd8353ec740 89
nhp: 2017-08-16 13:56:03.569675 send return 89 0 0 elapsed +0 00:00:00.000016
nhp: 2017-08-16 13:56:03.569680 send enter 0x7fd8353ec740 2
nhp: 2017-08-16 13:56:03.569690 send return 2 0 0 elapsed +0 00:00:00.000010
nhp: 2017-08-16 13:56:03.569695 send enter 0x7fd8353ec740 6
nhp: 2017-08-16 13:56:03.569709 send return 6 0 0 elapsed +0 00:00:00.000014
nhp: 2017-08-16 13:56:03.569714 send enter 0x7fd8353ec740 16384
nhp: 2017-08-16 13:56:03.569731 send return 16384 0 0 elapsed +0 00:00:00.000017
nhp: 2017-08-16 13:56:03.569737 send enter 0x7fd8353ec740 2
nhp: 2017-08-16 13:56:03.569742 send return 2 0 0 elapsed +0 00:00:00.000005
```

The wireshark dump from the client:

```
Authorization: AWS4-HMAC-SHA256 Credential=5QB12EADJ3HTU0PR5TQZ/20170816/us-east-1/s3/aws4_request, SignedHead
ers=content-encoding;content-type;date;host;x-amz-content-sha256;x-amz-date;x-amz-decoded-content-length;x-amz
-meta-blocksize;x-amz-meta-chunkidformat;x-amz-meta-chunkprefix;x-amz-meta-chunks;x-amz-meta-chunksize;x-amz-m
eta-chunktype;x-amz-meta-copynumber;x-amz-meta-dbid;x-amz-meta-dbname;x-amz-meta-filename;x-amz-meta-filesize;
x-amz-meta-filetype;x-amz-meta-incarnation;x-amz-meta-libraryname;x-amz-meta-node;x-amz-meta-opentime;x-amz-me
ta-sbtrequest;x-amz-meta-sbtretrycount;x-amz-meta-sbtversion;x-amz-meta-sessionid;x-amz-meta-status;x-amz-meta
-system;x-amz-meta-tickloc;x-amz-meta-user, Signature=f69993822618a847e0e101d9fff756471f3faed67808890e1b6fb536
c80c98c0

57
4000;chunk-signature=d61f8acc559fdf781581b6eda77de720360a2786f1d8449aa10654cf73997487

4000
.....
```

Then here's rgw detecting the chunked upload:

```
STREAMING-AWS4-HMAC-SHA256-PAYLOAD
2017-08-16 13:56:03.570349 7f4bdc4f5700 10 canonical request hash = 6a93f48bcc8d274117f48a50dc84c5cfd812b3fb7e
dc5be7d763459647423c62
2017-08-16 13:56:03.570358 7f4bdc4f5700 10 string to sign = AWS4-HMAC-SHA256
20170816T115603Z
20170816/us-east-1/s3/aws4_request
6a93f48bcc8d274117f48a50dc84c5cfd812b3fb7edc5be7d763459647423c62
2017-08-16 13:56:03.570387 7f4bdc4f5700 10 body content detected in multiple chunks
2017-08-16 13:56:03.570388 7f4bdc4f5700 10 aws4 seed signature ok... delaying v4 auth
```

```
2017-08-16 13:56:03.570503 7f4bdc4f5700 10 date_k    = aab84507d9ebbbeebff8608f793ad0cfc4c07b757c28829bc331a98
66ea04ee7
2017-08-16 13:56:03.570515 7f4bdc4f5700 10 region_k  = 301517aefddf7433ec49a49f2ba5a690e7a36a1c5176b4b02ffd324
ba6202d85
2017-08-16 13:56:03.570518 7f4bdc4f5700 10 service_k = 5ee8c521c9d09722fed63a9435c3b00c69b5ac29566cf95d8edcaac
9d037515c
2017-08-16 13:56:03.570528 7f4bdc4f5700 10 signing_k = ec979262faf07db88a9631f4fc75bb3954823ea84749c05fb883408
8a65cd841
2017-08-16 13:56:03.570552 7f4bdc4f5700 10 generated signature = f69993822618a847e0e101d9fff756471f3faed678088
90e1b6fb536c80c98c0
2017-08-16 13:56:03.570557 7f4bdc4f5700 15 string_to_sign=AWS4-HMAC-SHA256
20170816T115603Z
20170816/us-east-1/s3/aws4_request
6a93f48bcc8d274117f48a50dc84c5cfd812b3fb7edc5be7d763459647423c62
2017-08-16 13:56:03.570569 7f4bdc4f5700 15 server signature=f69993822618a847e0e101d9fff756471f3faed67808890e1b
6fb536c80c98c0
2017-08-16 13:56:03.570570 7f4bdc4f5700 15 client signature=f69993822618a847e0e101d9fff756471f3faed67808890e1b
6fb536c80c98c0
2017-08-16 13:56:03.570571 7f4bdc4f5700 15 compare=0
2017-08-16 13:56:03.570618 7f4bdc4f5700 10 date_k    = aab84507d9ebbbeebff8608f793ad0cfc4c07b757c28829bc331a98
66ea04ee7
2017-08-16 13:56:03.570627 7f4bdc4f5700 10 region_k  = 301517aefddf7433ec49a49f2ba5a690e7a36a1c5176b4b02ffd324
ba6202d85
2017-08-16 13:56:03.570631 7f4bdc4f5700 10 service_k = 5ee8c521c9d09722fed63a9435c3b00c69b5ac29566cf95d8edcaac
9d037515c
2017-08-16 13:56:03.570634 7f4bdc4f5700 10 signing_k = ec979262faf07db88a9631f4fc75bb3954823ea84749c05fb883408
8a65cd841
```

And failing on the first chunk:

```
2017-08-16 13:56:03.570755 7f4bdc4f5700  2 req 12:0.001047:s3:PUT /oracle-data-miropoto-1/file_chunk/0/SBTDB/u
nknown/2017-08-16/crap/ZyH3tH16jobK/0000000001:put_obj:executing
2017-08-16 13:56:03.570939 7f4bdc4f5700 20 parsed new chunk; signature=d61f8acc559fdf781581b6eda77de720360a278
6f1d8449aa10654cf73997487, data_length=87, data_starts_in_stream=91
2017-08-16 13:56:03.570952 7f4bdc4f5700 30 AWSv4ComplMulti: stream_pos_was=91, to_extract=87
2017-08-16 13:56:03.570954 7f4bdc4f5700 30 AWSv4ComplMulti: to_extract=87, data_len=10
2017-08-16 13:56:03.570962 7f4bdc4f5700 30 AWSv4ComplMulti: to_extract=77, received=77
2017-08-16 13:56:03.570964 7f4bdc4f5700 20 AWSv4ComplMulti: filled=87
2017-08-16 13:56:03.571023 7f4bdc4f5700 20 AWSv4ComplMulti: string_to_sign=
AWS4-HMAC-SHA256-PAYLOAD
20170816T115603Z
20170816/us-east-1/s3/aws4_request
f69993822618a847e0e101d9fff756471f3faed67808890e1b6fb536c80c98c0
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
3a79b501bbeaa09e83a00e71ecb07b3acd43acba55b805f393f543edf6c09376
2017-08-16 13:56:03.571054 7f4bdc4f5700 20 AWSv4ComplMulti: ERROR: chunk signature mismatch
2017-08-16 13:56:03.571058 7f4bdc4f5700 20 AWSv4ComplMulti: declared signature=d61f8acc559fdf781581b6eda77de72
0360a2786f1d8449aa10654cf73997487
2017-08-16 13:56:03.571059 7f4bdc4f5700 20 AWSv4ComplMulti: calculated signature=bd1a20b0df8a661c5a736bad4703d
fc0e0be28ac77c224fd71a9dd44d91697ce
```

#4 - 08/16/2017 02:10 PM - Radoslaw Zarzynski

It appears to me that something terribly wrong happened at the client side. Please take a look on the length-related headers of the failed request:

```
Content-Length: 1638400
x-amz-decoded-content-length: 1638400
```

These values are equal while the first one should account not only the real object size but also the chunking metadata. According to the AWS documentation:

> When transferring data in a series of chunks, you must use the Content-Length HTTP header to explicitly specify the total content length (object length in bytes plus metadata in each chunk). This will require you to pre-compute the total length of the payload including the metadata you will send in each chunk before starting your request. The x-amz-decoded-content-length header will contain the size of the object length in bytes.

Also the body begins with something that confuses me:

```
57
4000;chunk-signature=d61f8acc559fdf781581b6eda77de720360a2786f1d8449aa10654cf73997487
```

What does **57** here? Undoubtedly it's being interpreted by the std::strtoull in AWSv4ComplMulti::ChunkMeta::create_next as the hex-written chunk's length resulting in *data_length=87* in the log. However, the intended chunk metadata is sent in the next line.

#5 - 08/16/2017 03:25 PM - Dan van der Ster

Indeed that is strange. Also, is it normal to see **4000** twice before the data?
We are now testing this client with amazon, and it seems to use this same encoding, yet AWS is tolerating it and storing the objects.

#6 - 08/16/2017 03:42 PM - Dan van der Ster

Radoslaw Zarzynski wrote:

> What does **57** here?

87 is the length+1 of '4000;chunk-signature=d61f8acc559fdf781581b6eda77de720360a2786f1d8449aa10654cf73997487"

Is there a variant of the aws-chunked encoding whereby when Content-length==x-amz-decoded-content-length then the extra chunk metadata is counted just before each chunk?

#7 - 08/16/2017 07:28 PM - Radoslaw Zarzynski

> Is there a variant of the aws-chunked encoding whereby when Content-length==x-amz-decoded-content-length then the extra chunk metadata is counted just before each chunk?

I have never seen it before.

> We are now testing this client with amazon, and it seems to use this same encoding, yet AWS is tolerating it and storing the objects.

Maybe (just **speculating**!) AWS fallbacks to eg. single chunk mode (with unsigned payload) and stores the incoming HTTP body as is, without dechunking? Though, I can't recall any single sentence in the documentation claiming such behavior. Have you tried to retrieve the object back from S3?

What is also interesting is presence of extra chunking-related metadata in headers:

```
x-amz-meta-ChunkPrefix: file_chunk/0/SBTDB/unknown/2017-08-16/crap/ZyH3tH16jobK/
x-amz-meta-ChunkIdFormat: %010d
x-amz-meta-ChunkSize: 104857600
x-amz-meta-ChunkType: FileChunk
```

I had tried to google a bit for x-amz-meta-ChunkIdFormat but found nothing.

#8 - 08/16/2017 07:48 PM - Radoslaw Zarzynski

Wait a minute. It might be that we're having here aws-chunked payload wrapped with HTTP's chunked transfer encoding:

```
Content-Length: 1638400
...
```

```
Content-Encoding: aws-chunked
Transfer-Encoding: chunked
```

Though, the client is misbehaving as Content-Length must be absent in such situation. Quoting RFC 7230:

> A sender MUST NOT send a Content-Length header field in any message that contains a Transfer-Encoding header field.

Apparently AWS ignores the value of Content-Length and performs normal HTTP dechunking first. We might need to do the same, I think.

#9 - 08/16/2017 07:53 PM - Dan van der Ster

> Have you tried to retrieve the object back from S3?

I'll check again tomorrow but I believe the data was stored correctly. No chunk metadata within.

> What is also interesting is presence of extra chunking-related metadata in headers:

I believe those are just user-defined metadata for this backup client. It uses the term "Chunk" to refer the backup objects it creates... The default "Chunk" size is indeed 100MB. But this should have nothing to do with the aws chunks, afaict.

#10 - 08/16/2017 08:34 PM - Dan van der Ster

Radoslaw Zarzynski wrote:

> Wait a minute. It might be that we're having here aws-chunked payload wrapped with HTTP's chunked transfer encoding.

Found this via the wayback machine:

> If you don't want to calculate the length of the data including the metadata, you can instead use the Transfer-Encoding HTTP header with the

> value chunked. Amazon S3 supports this low-level alternative in the event the client you are using does not supports Transfer-Encoding.

https://web.archive.org/web/20140304132907/http://docs.aws.amazon.com/AmazonS3/latest/API/sigv4-streaming.html

#11 - 08/16/2017 09:03 PM - Radoslaw Zarzynski

- Related to Bug #21015: rgw: make HTTP dechunking compatible with Amazon S3 added

#12 - 08/16/2017 09:08 PM - Radoslaw Zarzynski

OK, I'm on it. As a stop gap solution you might consider to deploy RGW beside a proxy that would perform the HTTP TE dechunking.

#13 - 08/17/2017 10:10 AM - Dan van der Ster

Radoslaw Zarzynski wrote:

> OK, I'm on it. As a stop gap solution you might consider to deploy RGW beside a proxy that would perform the HTTP TE dechunking.

Excellent. With an apache proxy, rgw gets the decoded PUT, with correct Content-Length, and rgw correctly decodes the aws-chunked message. So now we have the PUT working -- but unfortunately it brings us to a new failure, some sort of client-side error "unsupported-algorithm". Digging into that now.

#14 - 08/18/2017 09:59 AM - Dan van der Ster

- File wireshark3.txt added

Attached is a wireshark dump of the response to a restore test.
We noticed the response has

```
Content-Encoding: aws-chunked
```

but then the content is **not** chunked. Is rgw persisting the Content-Encoding header then responding with how it was PUT into the storage? This would be incorrect, i guess...

#15 - 08/21/2017 09:46 AM - Radoslaw Zarzynski

CivetWeb, which is used as RadosGW's HTTP frontend, actually does support the chunked transfer encoding of incoming data. Though, the transform won't be applied if request has also Content-Length specified. Please take a look on the following line:

- https://github.com/ceph/civetweb/blob/wip-listen4/src/civetweb.c#L11974.

After rectifying the client's misbehavior, everything should work.

Additionally, following pull requests have been sent to give the priority to Transfer-Encoding, and thus tolerate such clients like AWS does:

- https://github.com/ceph/civetweb/pull/22,
- https://github.com/ceph/ceph/pull/17072.

#16 - 08/21/2017 09:55 AM - Abhishek Lekshmanan

- Status changed from New to Pending Backport

- Backport set to luminous

#17 - 08/21/2017 09:57 AM - Abhishek Lekshmanan

- Status changed from Pending Backport to In Progress

Ok, I see that the original patch doesn't solve the full issue yet, changing from backport -> in progress

#18 - 08/25/2017 02:43 PM - Dan van der Ster

I've opened a ticket for the aws-chunked GET bug: http://tracker.ceph.com/issues/21128

Files

ceph-client.rgw.cs3test.log.gz	51.1 KB	08/15/2017	Dan van der Ster
wireshark.txt	651 KB	08/16/2017	Dan van der Ster
sbtio_28681_140566666970880.log	247 KB	08/16/2017	Dan van der Ster
ceph-client.rgw.cs3test.log	242 KB	08/16/2017	Dan van der Ster
wireshark3.txt	4.05 KB	08/18/2017	Dan van der Ster