Network Working Group JM. Valin
Internet-Draft Mozilla Corporation
Updates: 6716 (if approved) K. Vos
Intended status: Standards Track vocTone
Expires: February 18, 2018 August 17, 2017

Updates to the Opus Audio Codec
draft-ietf-codec-opus-update-09

Abstract

This document addresses minor issues that were found in the specification of the Opus audio codec in RFC 6716. It updates the normative decoder implementation included in the appendix of RFC 6716. The changes fixes real and potential security-related issues, as well minor quality-related issues.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document addresses minor issues that were discovered in the reference implementation of the Opus codec. Unlike most IETF specifications, Opus is defined in RFC 6716 [RFC6716] in terms of a normative reference decoder implementation rather than from the associated text description. That RFC includes the reference decoder implementation as Appendix A. That's why only issues affecting the decoder are listed here. An up-to-date implementation of the Opus encoder can be found at <https://opus-codec.org/>.

Some of the changes in this document update normative behaviour in a way that requires new test vectors. The English text of the specification is unaffected, only the C implementation is. The updated specification remains fully compatible with the original specification.

Note: due to RFC formatting conventions, lines exceeding the column width in the patch are split using a backslash character. The backslashes at the end of a line and the white space at the beginning of the following line are not part of the patch. A properly formatted patch including all changes is available at <https://www.ietf.org/proceedings/98/slides/materials-98-codec-opus-update-00.patch> and has a SHA-1 hash of 029e3aa88fc342c91e67a21e7bfbc9458661cd5f.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3. Stereo State Reset in SILK

The reference implementation does not reinitialize the stereo state during a mode switch. The old stereo memory can produce a brief impulse (i.e. single sample) in the decoded audio. This can be fixed by changing silk/dec_API.c at line 72:

```
<CODE BEGINS>
     for( n = 0; n < DECODER_NUM_CHANNELS; n++ ) {
         ret  = silk_init_decoder( &channel_state[ n ] );
     }
+    silk_memset(&((silk_decoder *)decState)->sStereo, 0,
+                sizeof(((silk_decoder *)decState)->sStereo));
+    /* Not strictly needed, but it's cleaner that way */
+    ((silk_decoder *)decState)->prev_decode_only_middle = 0;

     return ret;
 }
<CODE ENDS>
```

This change affects the normative output of the decoder, but the amount of change is within the tolerance and too small to make the testvector check fail.

4. Parsing of the Opus Packet Padding

It was discovered that some invalid packets of very large size could trigger an out-of-bounds read in the Opus packet parsing code responsible for padding. This is due to an integer overflow if the signaled padding exceeds 2^31-1 bytes (the actual packet may be smaller). The code can be fixed by decrementing the (signed) len value, instead of incrementing a separate padding counter. This is done by applying the following changes at line 596 of src/opus_decoder.c:

```
<CODE BEGINS>
       /* Padding flag is bit 6 */
       if (ch&0x40)
       {
-         int padding=0;
          int p;
          do {
             if (len<=0)
                return OPUS_INVALID_PACKET;
             p = *data++;
             len--;
-            padding += p==255 ? 254: p;
+            len -= p==255 ? 254: p;
          } while (p==255);
-         len -= padding;
       }
<CODE ENDS>
```

This packet parsing issue is limited to reading memory up to about 60 kB beyond the compressed buffer. This can only be triggered by a compressed packet more than about 16 MB long, so it's not a problem for RTP. In theory, it could crash a file decoder (e.g. Opus in Ogg) if the memory just after the incoming packet is out-of-range, but our attempts to trigger such a crash in a production application built using an affected version of the Opus decoder failed.

5. Resampler buffer

The SILK resampler had the following issues:

1. The calls to memcpy() were using sizeof(opus_int32), but the type of the local buffer was opus_int16.

2. Because the size was wrong, this potentially allowed the source and destination regions of the memcpy() to overlap on the copy from "buf" to "buf". We believe that nSamplesIn (number of input samples) is at least fs_in_khZ (sampling rate in kHz), which is at least 8. Since RESAMPLER_ORDER_FIR_12 is only 8, that should not be a problem once the type size is fixed.

3. The size of the buffer used RESAMPLER_MAX_BATCH_SIZE_IN, but the data stored in it was actually twice the input batch size (nSamplesIn<<1).

The allocated buffers involved (buf and S->sFIR) are actually larger than they need to be for the batch size used, so no out-of-bounds read or write is possible. Therefore the bug cannot be exploited.

The code can be fixed by applying the following changes to line 78 of silk/resampler_private_IIR_FIR.c:

```
<CODE BEGINS>
 )
 {
     silk_resampler_state_struct *S = \
(silk_resampler_state_struct *)SS;
     opus_int32 nSamplesIn;
     opus_int32 max_index_Q16, index_increment_Q16;
-    opus_int16 buf[ RESAMPLER_MAX_BATCH_SIZE_IN + \
RESAMPLER_ORDER_FIR_12 ];
+    opus_int16 buf[ 2*RESAMPLER_MAX_BATCH_SIZE_IN + \
RESAMPLER_ORDER_FIR_12 ];

     /* Copy buffered samples to start of buffer */
-    silk_memcpy( buf, S->sFIR, RESAMPLER_ORDER_FIR_12 \
* sizeof( opus_int32 ) );
+    silk_memcpy( buf, S->sFIR, RESAMPLER_ORDER_FIR_12 \
* sizeof( opus_int16 ) );

     /* Iterate over blocks of frameSizeIn input samples */
     index_increment_Q16 = S->invRatio_Q16;
     while( 1 ) {
         nSamplesIn = silk_min( inLen, S->batchSize );

         /* Upsample 2x */
         silk_resampler_private_up2_HQ( S->sIIR, &buf[ \
RESAMPLER_ORDER_FIR_12 ], in, nSamplesIn );

         max_index_Q16 = silk_LSHIFT32( nSamplesIn, 16 + 1 \
);         /* + 1 because 2x upsampling */
         out = silk_resampler_private_IIR_FIR_INTERPOL( out, \
buf, max_index_Q16, index_increment_Q16 );
         in += nSamplesIn;
         inLen -= nSamplesIn;

         if( inLen > 0 ) {
             /* More iterations to do; copy last part of \
filtered signal to beginning of buffer */
-            silk_memcpy( buf, &buf[ nSamplesIn << 1 ], \
RESAMPLER_ORDER_FIR_12 * sizeof( opus_int32 ) );
+            silk_memmove( buf, &buf[ nSamplesIn << 1 ], \
RESAMPLER_ORDER_FIR_12 * sizeof( opus_int16 ) );
         } else {
             break;
         }
     }
```

```
     /* Copy last part of filtered signal to the state for \
the next call */
-    silk_memcpy( S->sFIR, &buf[ nSamplesIn << 1 ], \
RESAMPLER_ORDER_FIR_12 * sizeof( opus_int32 ) );
+    silk_memcpy( S->sFIR, &buf[ nSamplesIn << 1 ], \
RESAMPLER_ORDER_FIR_12 * sizeof( opus_int16 ) );
 }
<CODE ENDS>
```

6. Integer wrap-around in inverse gain computation

It was discovered through decoder fuzzing that some bitstreams could produce integer values exceeding 32-bits in LPC_inverse_pred_gain_QA(), causing a wrap-around. Although the authors are not aware of any way to exploit the bug, the C standard considers the behavior as undefined. The following patch to line 87 of silk/LPC_inv_pred_gain.c detects values that do not fit in a 32-bit integer and considers the corresponding filters unstable:

```
<CODE BEGINS>
         /* Update AR coefficient */
         for( n = 0; n < k; n++ ) {
-            tmp_QA = Aold_QA[ n ] - MUL32_FRAC_Q( \
Aold_QA[ k - n - 1 ], rc_Q31, 31 );
-            Anew_QA[ n ] = MUL32_FRAC_Q( tmp_QA, rc_mult2 , mult2Q );
+            opus_int64 tmp64;
+            tmp_QA = silk_SUB_SAT32( Aold_QA[ n ], MUL32_FRAC_Q( \
Aold_QA[ k - n - 1 ], rc_Q31, 31 ) );
+            tmp64 = silk_RSHIFT_ROUND64( silk_SMULL( tmp_QA, \
rc_mult2 ), mult2Q);
+            if( tmp64 > silk_int32_MAX || tmp64 < silk_int32_MIN ) {
+               return 0;
+            }
+            Anew_QA[ n ] = ( opus_int32 )tmp64;
         }
<CODE ENDS>
```

7. Integer wrap-around in LSF decoding

It was discovered -- also from decoder fuzzing -- that an integer wrap-around could occur when decoding bitstreams with extremely large values for the high LSF parameters. The end result of the wrap-around is an illegal read access on the stack, which the authors do not believe is exploitable but should nonetheless be fixed. The following patch to line 137 of silk/NLSF_stabilize.c prevents the problem:

```
<CODE BEGINS>
           /* Keep delta_min distance between the NLSFs */
         for( i = 1; i < L; i++ )
-            NLSF_Q15[i] = silk_max_int( NLSF_Q15[i], \
NLSF_Q15[i-1] + NDeltaMin_Q15[i] );
+            NLSF_Q15[i] = silk_max_int( NLSF_Q15[i], \
silk_ADD_SAT16( NLSF_Q15[i-1], NDeltaMin_Q15[i] ) );

         /* Last NLSF should be no higher than 1 - NDeltaMin[L] */
<CODE ENDS>
```

8. Cap on Band Energy

On extreme bit-streams, it is possible for log-domain band energy levels to exceed the maximum single-precision floating point value once converted to a linear scale. This would later cause the decoded values to be NaN (not a number), possibly causing problems in the software using the PCM values. This can be avoided with the following patch to line 552 of celt/quant_bands.c:

```
<CODE BEGINS>
       {
          opus_val16 lg = ADD16(oldEBands[i+c*m->nbEBands],
                          SHL16((opus_val16)eMeans[i],6));
+         lg = MIN32(QCONST32(32.f, 16), lg);
          eBands[i+c*m->nbEBands] = PSHR32(celt_exp2(lg),4);
       }
       for (;i<m->nbEBands;i++)
<CODE ENDS>
```

9. Hybrid Folding

When encoding in hybrid mode at low bitrate, we sometimes only have enough bits to code a single CELT band (8 - 9.6 kHz). When that happens, the second band (CELT band 18, from 9.6 to 12 kHz) cannot use folding because it is wider than the amount already coded, and falls back to white noise. Because it can also happen on transients (e.g. stops), it can cause audible pre-echo.

To address the issue, we change the folding behavior so that it is never forced to fall back to LCG due to the first band not containing enough coefficients to fold onto the second band. This is achieved by simply repeating part of the first band in the folding of the second band. This changes the code in celt/bands.c around line 1237:

```
<CODE BEGINS>
          b = 0;
       }

-      if (resynth && M*eBands[i]-N >= M*eBands[start] && \
(update_lowband || lowband_offset==0))
+      if (resynth && (M*eBands[i]-N >= M*eBands[start] || \
i==start+1) && (update_lowband || lowband_offset==0))
             lowband_offset = i;

+      if (i == start+1)
+      {
+         int n1, n2;
+         int offset;
+         n1 = M*(eBands[start+1]-eBands[start]);
+         n2 = M*(eBands[start+2]-eBands[start+1]);
+         offset = M*eBands[start];
+         /* Duplicate enough of the first band folding data to \
be able to fold the second band.
+            Copies no data for CELT-only mode. */
+         OPUS_COPY(&norm[offset+n1], &norm[offset+2*n1 - n2], n2-n1);
+         if (C==2)
+            OPUS_COPY(&norm2[offset+n1], &norm2[offset+2*n1 - n2], \
n2-n1);
+      }
+
       tf_change = tf_res[i];
       if (i>=m->effEBands)
       {
<CODE ENDS>
```

as well as line 1260:

```
<CODE BEGINS>
          fold_start = lowband_offset;
          while(M*eBands[--fold_start] > effective_lowband);
          fold_end = lowband_offset-1;
-         while(M*eBands[++fold_end] < effective_lowband+N);
+         while(++fold_end < i && M*eBands[fold_end] < \
effective_lowband+N);
          x_cm = y_cm = 0;
          fold_i = fold_start; do {
            x_cm |= collapse_masks[fold_i*C+0];

<CODE ENDS>
```

The fix does not impact compatibility, because the improvement does not depend on the encoder doing anything special. There is also no

reasonable way for an encoder to use the original behavior to improve quality over the proposed change.

10. Downmix to Mono

The last issue is not strictly a bug, but it is an issue that has been reported when downmixing an Opus decoded stream to mono, whether this is done inside the decoder or as a post-processing step on the stereo decoder output. Opus intensity stereo allows optionally coding the two channels 180-degrees out of phase on a per-band basis. This provides better stereo quality than forcing the two channels to be in phase, but when the output is downmixed to mono, the energy in the affected bands is cancelled sometimes resulting in audible artifacts.

As a work-around for this issue, the decoder MAY choose not to apply the 180-degree phase shift. This can be useful when downmixing to mono inside or outside of the decoder (e.g. user-controllable).

11. New Test Vectors

Changes in Section 9 and Section 10 have sufficient impact on the testvectors to make them fail. For this reason, this document also updates the Opus test vectors. The new test vectors now include two decoded outputs for the same bitstream. The outputs with suffix 'm' do not apply the CELT 180-degree phase shift as allowed in Section 10, while the outputs without the suffix do. An implementation is compliant as long as it passes either set of vectors.

Any Opus implementation that passes either the original test vectors from RFC 6716 [RFC6716] or one of the new sets of test vectors is compliant with the Opus specification. However, newer implementations SHOULD be based on the new test vectors rather than the old ones.

The new test vectors are located at <https://www.ietf.org/proceedings/98/slides/materials-98-codec-opus-newvectors-00.tar.gz>. The SHA-1 hashes of the test vectors are:

```
e49b2862ceec7324790ed8019eb9744596d5be01  testvector01.bit
b809795ae1bcd606049d76de4ad24236257135e0  testvector02.bit
e0c4ecaeab44d35a2f5b6575cd996848e5ee2acc  testvector03.bit
a0f870cbe14ebb71fa9066ef3ee96e59c9a75187  testvector04.bit
9b3d92b48b965dfe9edf7b8a85edd4309f8cf7c8  testvector05.bit
28e66769ab17e17f72875283c14b19690cbc4e57  testvector06.bit
bacf467be3215fc7ec288f29e2477de1192947a6  testvector07.bit
ddbe08b688bbf934071f3893cd0030ce48dba12f  testvector08.bit
3932d9d61944dab1201645b8eeaad595d5705ecb  testvector09.bit
521eb2a1e0cc9c31b8b740673307c2d3b10c1900  testvector10.bit
6bc8f3146fcb96450c901b16c3d464ccdf4d5d96  testvector11.bit
338c3f1b4b97226bc60bc41038becbc6de06b28f  testvector12.bit
f5ef93884da6a814d311027918e9afc6f2e5c2c8  testvector01.dec
48ac1ff1995250a756e1e17bd32acefa8cd2b820  testvector02.dec
d15567e919db2d0e818727092c0af8dd9df23c95  testvector03.dec
1249dd28f5bd1e39a66fd6d99449dca7a8316342  testvector04.dec
b85675d81deef84a112c466cdff3b7aaa1d2fc76  testvector05.dec
55f0b191e90bfa6f98b50d01a64b44255cb4813e  testvector06.dec
61e8b357ab090b1801eeb578a28a6ae935e25b7b  testvector07.dec
a58539ee5321453b2ddf4c0f2500e856b3966862  testvector08.dec
bb96aad2cde188555862b7bbb3af6133851ef8f4  testvector09.dec
1b6cdf0413ac9965b16184b1bea129b5c0b2a37a  testvector10.dec
b1fff72b74666e3027801b29dbc48b31f80dee0d  testvector11.dec
98e09bbafed329e341c3b4052e9c4ba5fc83f9b1  testvector12.dec
1e7d984ea3fbb16ba998aea761f4893fbdb30157  testvector01m.dec
48ac1ff1995250a756e1e17bd32acefa8cd2b820  testvector02m.dec
d15567e919db2d0e818727092c0af8dd9df23c95  testvector03m.dec
1249dd28f5bd1e39a66fd6d99449dca7a8316342  testvector04m.dec
d70b0bad431e7d463bc3da49bd2d49f1c6d0a530  testvector05m.dec
6ac1648c3174c95fada565161a6c78bdbe59c77d  testvector06m.dec
fc5e2f709693738324fb4c8bdc0dad6dda04e713  testvector07m.dec
aad2ba397bf1b6a18e8e09b50e4b19627d479f00  testvector08m.dec
6feb7a7b9d7cdc1383baf8d5739e2a514bd0ba08  testvector09m.dec
1b6cdf0413ac9965b16184b1bea129b5c0b2a37a  testvector10m.dec
fd3d3a7b0dfbdab98d37ed9aa04b659b9fefbd18  testvector11m.dec
98e09bbafed329e341c3b4052e9c4ba5fc83f9b1  testvector12m.dec
```

Note that the decoder input bitstream files (.bit) are unchanged.

12. Security Considerations

This document fixes two security issues reported on Opus and that affect the reference implementation in RFC 6716 [RFC6716]: CVE-2013-0899 <https://nvd.nist.gov/vuln/detail/CVE-2013-0899> and CVE-2017-0381 <https://nvd.nist.gov/vuln/detail/CVE-2017-0381>. CVE-2013-0899 is fixed by Section 4 and could theoretically cause an information leak, but the leaked information would at the very least

go through the decoder process before being accessible to the attacker. Also, the bug can only be triggered by Opus packets at least 24 MB in size. CVE-2017-0381 is fixed by Section 7 and can only result in a 16-bit out-of-bounds read to a fixed location 256 bytes before a constant table. That location would normally be part of an executable's read-only data segment, but if that is not the case, the bug could at worst results in either a crash or the leakage of 16 bits of information from that fixed memory location (if the attacker has access to the decoded output). Despite the claims of the CVE, the bug cannot results in arbitrary code execution. Beyond the two fixed CVEs, this document adds no new security considerations on top of RFC 6716 [RFC6716].

13. IANA Considerations

This document makes no request of IANA.

Note to RFC Editor: this section may be removed on publication as an RFC.

14. Acknowledgements

We would like to thank Juri Aedla for reporting the issue with the parsing of the Opus padding. Thanks to Felicia Lim for reporting the LSF integer overflow issue. Also, thanks to Tina le Grand, Jonathan Lennox, and Mark Harris for their feedback on this document.

15. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC6716] Valin, JM., Vos, K., and T. Terriberry, "Definition of the Opus Audio Codec", RFC 6716, DOI 10.17487/RFC6716, September 2012, <https://www.rfc-editor.org/info/rfc6716>.

Authors' Addresses

Jean-Marc Valin
Mozilla Corporation
331 E. Evelyn Avenue
Mountain View, CA 94041
USA

Phone: +1 650 903-0800
Email: jmvalin@jmvalin.ca

Koen Vos
vocTone

Email: koenvos74@gmail.com