Integer overflows vulnerabilities in EDK2 Tiano decompression algorithm

Table of Contents

Contact information

If you have questions related to vulnerabilities from this security advisory, you can contact:

Bazhaniuk Oleksandr (alex@eclypsium.com), Perez Federico (perezfederico@unc.edu.ar), Bulygin Yuriy (yuriy@eclypsium.com)

Abstract

In number of cases integer overflows causes heap overflow vulnerabilities in decompression algorithm of EDK2 Tiano reference implementation.

Timeline

Reported at 18th of October 2017

Acknowledgements

Reported by Oleksandr Bazhaniuk, Perez Federico, Bulygin Yuriy.

Introduction

The following report summarizes details of the vulnerabilities found in the EDK2 implementation of Tiano decompression algorithm.

Vulnerabilities affect:

- IntelFrameworkModulePkg: https://github.com/tianocore/edk2/blob/master/IntelFrameworkModulePkg/Library/BaseUefiTianoCustomDecompressLib/BaseUefiTianoCustomDecompressLib.c
- Base Tools
- DuetPkg: https://github.com/tianocore/edk2/blob/master/DuetPkg/EfiLdr/TianoDecompress.c
- Other UEFI packages which use Tiano decompression algorithm

Vulnerabilities #1

Integer overflow that cause heap overflow.

To reproduce use EDK2 Tiano tools:

```
./TianoCompress.bin -d sect01_yrfa.gz
```

Trying to decompress this file with the Tiano Compressor standalone executable from EDK2 triggers a SEGFAULT. In general this vulnerability affects all unpatched implementation of decompression algorithm.

Analysis of vulnerability #1

Running the program through GDB, and inspecting the variables in the place of the crash...

```
Reading symbols from ./TianoCompress...done.
(gdb) r
Starting program:
/media/fede/Storage/Chipsec/edk2/BaseTools/Source/C/bin/TianoCompress -d
../../../../../ImagesForBugDecompress/sect01_yrfa.gz

Program received signal SIGSEGV, Segmentation fault.
0x000055555555827b in Decode (Sd=0x55555575f010) at TianoCompress.c:2622
2622            Sd->mDstBase[Sd->mOutBuf++] = Sd->mDstBase[DataIdx++];
(gdb) info locals
BytesRemain = 121
DataIdx = 4294543956
CharC = 122
```

There is an overflow in the `DataIdx` value, causing a memory corruption when trying to access a buffer with index which is overflowed (`4294543956`):

Overflow of `DataIdx` happens in line:

```
DataIdx     = Sd->mOutBuf - DecodeP (Sd) - 1;
```

Memory corruption happens in line:

```
Sd->mDstBase[Sd->mOutBuf++] = Sd->mDstBase[DataIdx++];
```

Attacker is controlling `DataIdx` which is allowing attacker has arbitrary read primitive.

Also Attacker is controlling `CharC`, as result controlling `BytesRemain` counter (in boundaries of maximum value in UINT16). It's cause heap overflow in:

```
Sd->mDstBase[Sd->mOutBuf++] = Sd->mDstBase[DataIdx++];
```

Mitigation of vulnerability #1

Solution was to add sanity check for integer overflow and an error return value to let know the parent function that there was a fault and discard the "decompressed data":

```
DataIdx     = Sd->mOutBuf - DecodeP (Sd) - 1;

//If algorithm is not correct, there is an overflow possibility
if (DataIdx > Sd->mOrigSize) {
        Sd→mBadAlgorithm = 1;
        return;
   }
```

Then, at the output, the `mBadAlgorithm` value is checked and properly managed.

```
if (Sd->mBadTableFlag != 0 || Sd->mBadAlgorithm != 0) {
        Status = EFI_INVALID_PARAMETER;
 }
```

Vulnerabilities #2

When the file to decompress has a wrong format, or has the wrong type (EFI i.e.) there is an heap overflow in the `mPTTable` buffer in the `SCRATCH_DATA` structure.

To reproduce use EDK2 Tiano tools (with patch for vulnerability #1):

```
./TianoCompress.bin -d sect01_yrfa.gz
```

Trying to decompress this file with the Tiano Compressor standalone executable from EDK2 triggers a SEGFAULT. In general this vulnerability affects all unpatched implementation of decompression algorithm.

Analysis of vulnerability #2

Running BaseTools tool program with Valgrind to detect this vulnerability:

```
==22847== Command: ./TianoCompress -d ../../../../../ImagesForBugDecompress/sect01_yrfa.gz
==22847==
==22847== Invalid write of size 2
==22847==    at 0x10BAB3: MakeTable (TianoCompress.c:2236)
==22847==    by 0x10BD87: ReadPTLen (TianoCompress.c:2407)
==22847==    by 0x10C1F1: DecodeC (TianoCompress.c:2538)
==22847==    by 0x10C21F: Decode (TianoCompress.c:2595)
==22847==    by 0x10C398: Decompress (TianoCompress.c:2726)
==22847==    by 0x1091F2: main (TianoCompress.c:1997)
==22847==  Address 0x5204480 is 0 bytes after a block of size 13,376 alloc'd
==22847==    at 0x4C2DB2F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==22847==    by 0x1090DA: main (TianoCompress.c:1890)
==22847==
==22847== Invalid write of size 2
==22847==    at 0x10BAA8: MakeTable (TianoCompress.c:2237)
==22847==    by 0x10BD87: ReadPTLen (TianoCompress.c:2407)
==22847==    by 0x10C1F1: DecodeC (TianoCompress.c:2538)
==22847==    by 0x10C21F: Decode (TianoCompress.c:2595)
==22847==    by 0x10C398: Decompress (TianoCompress.c:2726)
==22847==    by 0x1091F2: main (TianoCompress.c:1997)
==22847==  Address 0x5204486 is 6 bytes after a block of size 13,376 alloc'd
==22847==    at 0x4C2DB2F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==22847==    by 0x1090DA: main (TianoCompress.c:1890)
==22847==
==22847== Invalid read of size 1
==22847==    at 0x10C27B: Decode (TianoCompress.c:2622)
==22847==    by 0x10C398: Decompress (TianoCompress.c:2726)
==22847==    by 0x1091F2: main (TianoCompress.c:1997)
==22847==  Address 0x10519fd23 is not stack'd, malloc'd or (recently) free'd
==22847==
==22847==
==22847== Process terminating with default action of signal 11 (SIGSEGV)
==22847==  Access not within mapped region at address 0x10519FD23
==22847==    at 0x10C27B: Decode (TianoCompress.c:2622)
==22847==    by 0x10C398: Decompress (TianoCompress.c:2726)
==22847==    by 0x1091F2: main (TianoCompress.c:1997)
```

There are an invalid writes in the `MakeTable` function and then in another iteration of the `Decode` function the memory corruption appears.

Inspecting the function there is interesting places where the input `Table` is modified:

Code from MakeTable function:

```
NextCode = (UINT16) (Start[Len] + Weight[Len]);

    if (Len <= TableBits) {

      for (Index = Start[Len]; Index < NextCode; Index++) {
        Table[Index] = Char;
      }
```

It doesn't check that `Index` value in boundaries of `Table` buffer. Attacker controlling `Index` and it is allowing attacker to have heap overflow in the `Table` buffer. In this scenario attacker is controlling "offset" and "data" during overflow which is often can be exploited to arbitrary code execution attack.

Mitigation of vulnerability #2

Solution was to add sanity check for integer overflow:

```
UINT16  TableSize;
```

```
TableSize = (UINT16) (1U << TableBits);

for (Index = Start[Len]; Index < NextCode; Index++) {
         if(Index >= TableSize)
         {
           //Set error here or...
           return (UINT16) BAD_TABLE;
         }
          Table[Index] = Char;
        }
```

References

- EDK2 Files:
  https://github.com/tianocore/edk2/tree/master/BaseTools/Source/C/TianoCompress

- For a complete fix on these vulnerabilities and other non-critical fixes check:
  https://github.com/chipsec/chipsec/pull/294

  Files to reproduce vulnerabilities can find in attachment to
  https://github.com/chipsec/chipsec/issues/269:

  P11-B2.zip
  3440a02.zip
  sect01_yrfa.zip