# Issue 1

Within in TianoCompress.c (BaseTools/Source/C/TianoCompress) from github/tianocore/edk2. File does not conform to UEFI Standards. This allows the binary to process malformed files with TianoCompress. (within main fuction)

UEFI Standard:

if (SrcSize < 8)   
 { return EFI\_INVALID\_PARAMETER; }   
CompSize = Src[0] + (Src[1] << 8) + (Src[2] << 16) + (Src[3] << 24);   
OrigSize = Src[4] + (Src[5] << 8) + (Src[6] << 16) + (Src[7] << 24);   
if (SrcSize < CompSize + 8)   
{ return EFI\_INVALID\_PARAMETER; }

Root Cause

Compsize is read in as a header value and not properly validated

Example

Output causing crash: within crash1.bin.   
  
Suggested Fix

Ensure SrcSize validation is included. Include CompSize validation. See issue 2.  
  
Issue 2

Within BaseUefiDecompressLib.c (MdePkg/Library/BaseUefiDecompressLib) from github/tianocore/edk2 UefiDecompress function, TianoCompress.c (within basetools) main function, and the UEFI Standard Decompress function, malformed header input is not sufficiently validated.

Root Cause

if (SrcSize < CompSize +8) and (SrcSize < 8), does not check for the edge case where adding 8 to CompSize causes the value to be less than SrcSize.

Example

Output causing Heap Based Buffer Overflow within crash1.bin. Within this example CompSize is greater than Source Size. As the header gives CompSize a value of FFFFFFFF. The input makes it past the check and later crashes within the Fillbuf function. The crash event is a result of mInBuf being larger than the size of SrcBase (the source file) causing data beyond the source size to be accessed. See below:

if (Sd->mCompSize > 0) {  
 //  
 // Get 1 byte into SubBitBuf  
 //  
 Sd->mCompSize --;  
 Sd->mSubBitBuf = 0;  
 Sd->mSubBitBuf = = Sd->mSrcBase[Sd->mInBuf ++];   
 Sd->mBitCount = 8;

Suggested Fix

Add additional check to ensure that Compressed Size +8 is greater than 8. See Below:

if (SrcSize < 8)   
 { return EFI\_INVALID\_PARAMETER; }   
CompSize = Src[0] + (Src[1] << 8) + (Src[2] << 16) + (Src[3] << 24);   
OrigSize = Src[4] + (Src[5] << 8) + (Src[6] << 16) + (Src[7] << 24);   
if (SrcSize < CompSize + 8 || (CompSize + 8) < 8)   
 { return EFI\_INVALID\_PARAMETER; }

Issue 3

Within BaseUefiDecompressLib.c, TianoCompress.c, and UEFI specification. No bounds checking within MakeTable Function which causes stack based buffer overflow

Root Cause

BitLen[i] can be greater than the size of Count (which is 17), causing Count to manipulate data on the stack.

UINT16 Count[17];  
UINT16 Weight[17];  
UINT16 Start[18];  
UINT16 \*p;  
UINT16 k;  
UINT16 i;  
UINT16 Len;  
UINT16 Char;  
UINT16 JuBits;  
UINT16 Avail;  
UINT16 NextCode;  
UINT16 Mask;

for (i = 1; i <= 16; i ++) {  
 Count[i] = 0;  
}

for (i = 0; i < NumOfChar; i++) {  
 Count[BitLen[i]]++;  
}

Example

Output causing Stack based Overflow within crash2.bin.

Suggested Fix

Add a check to ensure that BitLen[i] is < 16 and returning (UINT16) BAD\_TABLE if found.

for (i = 0; i < NumOfChar; i++) {  
 if (BitLen[i] > 16){  
 return (UINT16) BAD\_TABLE;  
 }  
 Count[BitLen[i]]++;  
}

Issue 4

Within BaseUefiDecompressLib.c, TianoCompress.c, and UEFI specification. No bounds checking within MakeTable Function which causes heap based buffer overflow

Root Cause

There is no bounds checking within the MakeTable function when writing values to the Table buffer. Function Maketable:

UINT16  
MakeTable (  
 IN SCRATCH\_DATA \*Sd,  
 IN UINT16 NumOfChar,  
 IN UINT8 \*BitLen,  
 IN UINT16 TableBits,  
 OUT UINT16 \*Table  
 )

is called with two instances :

return MakeTable (Sd, nn, Sd->mPTLen, 8, Sd->mPTTable);

and

MakeTable (Sd, NC, Sd->mCLen, 12, Sd->mCTable);

The size of both are denoted:

UINT16 mCTable[4096];

UINT16 mPTTable[256];

Within the code Table[Index] can be larger than either size of mPTTable or mCTable.

for (Char = 0; Char < NumOfChar; Char++) {

Len = BitLen[Char];

if (Len == 0 || Len >= 17) {

continue;

}

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

if (Len <= TableBits) {

for (Index = Start[Len]; Index < NextCode; Index++) {

Table[Index] = Char;

}

Example

Output causing Stack based Overflow within crash3.bin.

Suggested Fix

Add a check to ensure that Table[Index] is < sizeof(Table) and returning (UINT16) BAD\_TABLE if found.

for (Index = Start[Len]; Index < NextCode; Index++) {  
 if ( Index > (UINT16) (1U << TableBits)) {  
 return (UINT16) BAD\_TABLE;

}

Table[Index] = Char;

}

Issue 5

Within BaseUefiDecompressLib.c, TianoCompress.c, and UEFI specification. No bounds checking within Decode Function which causes heap based buffer overflow

Root Cause

Function Decode else condition does not validate that return value Sd->mOutBuf – DecodeP (Sd) -1, which is DataIdx, does not read and write outside of buffer Sd->mDstBase within BytesRemain loop:

// Process a Pointer

//

CharC = (UINT16) (CharC - (BIT8 - THRESHOLD));

//

// Get string length

//

BytesRemain = CharC;

//

// Locate string position

//

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

//

// Write BytesRemain of bytes into mDstBase

//

BytesRemain--;

while ((INT16) (BytesRemain) >= 0) {

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

if (Sd->mOutBuf >= Sd->mOrigSize) {

goto Done;

}

BytesRemain--;

}

Example

Output causing heap based Overflow within crash4.bin.

## Suggested Fix

The error was reading beyond the Sd->mDstBase[DataIdx++] buffer. mDstBase is the original file buffer being decoded. Therefore one way to fix the error is to bounds check before entering the loop as noted below. Another way to fix the problem (not explored) would be to see why DataIdx is being returned out of bounds for the buffer.

// Write BytesRemain of bytes into mDstBase

//

BytesRemain--;

if ( (DataIdx > Sd->mOrigSize) || (Sd->mOutBuf + BytesRemain > Sd->mOrigSize) || (DataIdx + BytesRemain > Sd->mOrigSize) ){

goto Done

}

while ((INT16) (BytesRemain) >= 0) {

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

if (Sd->mOutBuf >= Sd->mOrigSize) {

goto Done;

}

BytesRemain--;

}