Introduction

Libebml is an open source library written in C++ that parses EBML files. EBML (Extensible Binary Markup Language) is a simplified binary extension of xml.

About vulnerability

CVE-2015-8791 is a buffer over-read vulnerability in Libebml EbmlElement.cpp function ReadCodedSizeValue(). The vulnerability was in the Libebml until version 1.3.3. By Wikipedia definition Buffer over-read is anomaly where a program while reading data from buffer overruns the buffer’s boundary and reads adjacent memory[1]. In this case it was caused by the fact that the function didn’t have buffer checking coded in so when the function is called with buffer size value that exceeds the actual size of buffer then data outside of buffer is also read. This might lead to leak of confidential data from heap. The attack could be executed remotely, but user interaction is required in order for it to work. Vulnerability was made possible because the code was written in C++ which has no protection against accessing data in any parts of virtual memory and by the fact that the person working on that code didn’t write in checks to ensure that no data is read outside of buffer.

Function with vulnerability:

```cpp
1. uint64 ReadCodedSizeValue(const binary * InBuffer, uint32 & BufferSize, uint64 & SizeUnknown)
2. {
3.   binary SizeBitMask = 1 << 7;
4.   uint64 Result = 0x7F;
5.   unsigned int SizeIdx, PossibleSizeLength = 0;
6.   binary PossibleSize[8];
7.   memset(PossibleSize, 0, 8);
8.   SizeUnknown = 0x7F; // the last bit is discarded when computing the size
9.   for (SizeIdx = 0; SizeIdx < BufferSize && SizeIdx < 8; SizeIdx++) {
10.      if (InBuffer[0] & (SizeBitMask >> SizeIdx)) {
11.         // ID found
12.         PossibleSizeLength = SizeIdx + 1;
13.         SizeBitMask >>= SizeIdx;
14.         for (SizeIdx = 0; SizeIdx < PossibleSizeLength; SizeIdx++) {
15.             PossibleSize[SizeIdx] = InBuffer[SizeIdx];
16.         }
17.      }
18.      for (SizeIdx = 0; SizeIdx < PossibleSizeLength - 1; SizeIdx++) {
19.         Result <<= 7;
20.         Result |= 0xFF;
21.      }
22.      Result = 0;
23.      Result |= PossibleSize[0] & ~SizeBitMask;
24.      for (unsigned int i = 1; i < PossibleSizeLength; i++) {
25.         Result <<= 8;
26.         Result |= PossibleSize[i];
27.      }
```
Vulnerability fix

Function where vulnerability has been fixed:

```c
1. uint64 ReadCodedSizeValue(const binary * InBuffer, uint32 & BufferSize, uint64 & SizeUnknown)
2. {
3.     binary SizeBitMask = 1 << 7;
4.     uint64 Result = 0x7F;
5.     unsigned int SizeIdx, PossibleSizeLength = 0;
6.     binary PossibleSize[8];
7.     memset(PossibleSize, 0, 8);
8.     SizeUnknown = 0x7F; // the last bit is discarded when computing the size
9.     for (SizeIdx = 0; SizeIdx < BufferSize && SizeIdx < 8; SizeIdx++) {
10.        if (InBuffer[0] & (SizeBitMask >> SizeIdx)) {
11.            // ID found
12.                PossibleSizeLength = SizeIdx + 1;
13.                SizeBitMask >>= SizeIdx;
14.            } // ID found
15.        }
16.     // Guard against invalid memory accesses with incomplete IDs.
17.     if (PossibleSizeLength > BufferSize) break;
18. }
19. for (SizeIdx = 0; SizeIdx < PossibleSizeLength; SizeIdx++) {
20.    PossibleSize[SizeIdx] = InBuffer[SizeIdx];
21. }
22. for (SizeIdx = 0; SizeIdx < PossibleSizeLength - 1; SizeIdx++) {
23.    Result <<= 7;
24.    Result |= 0xFF;
25. }
26. Result = 0;
27. Result |= PossibleSize[0] & ~SizeBitMask;
28. for (unsigned int i = 1; i < PossibleSizeLength; i++) {
29.    Result <<= 8;
30.    Result |= PossibleSize[i];
31. }
32. BufferSize = PossibleSizeLength;
33. return Result;
34. }
35. SizeUnknown <<= 7;
36. SizeUnknown |= 0xFF;
37. }
```
The fix for this vulnerability was quite simple, consisting of only two lines of code:

```c
1. if (PossibleSizeLength > BufferSize)
2.     break;
```

In every iteration this code checks whether or not the next byte read is inside of buffer. If that’s not the case the loop will be broken and 0 will be returned instead of read data.
References

1 https://en.wikipedia.org/wiki/Buffer_over-read
2 https://cwe.mitre.org/data/definitions/126.html
3 https://github.com/Matroska-Org/libebml/blob/24e5cd7c666b1ddd85619d60486db0a5481c1b90/src/EbmlElement.cpp
5 http://matroska-org.github.io/libebml/