2016 年 5 月 11 日，星期三

TALOS 发现多个 7-ZIP 漏洞

作者：Marcin Noga 和 Jaeson Schultz。

7-Zip 是一款开源文件存档应用，其主要特性包括：可选 AES-256 加密；支持大文件；能够兼容“任何压缩、转换或加密方法”。最近，思科 Talos 团队在 7-Zip 中发现了多个可被利用的漏洞。这些类型的漏洞格外值得关注，因为厂商可能不会意识到他们使用了存在漏洞的代码库。举例来说，如果厂商将此应用集成到安全设备或防病毒产品，就会特别令人担忧。目前，所有主要的平台都支持 7-Zip，该应用已成为当今最受欢迎的常用存档工具之一。 如果说出受影响的产品和设备的数量，可能会令人感到吃惊。

TALOS-CAN-0094，越界读取漏洞，[CVE-2016-2335]

7-Zip 处理通用光盘格式 (UDF) 文件的方式存在越界读取漏洞。UDF 文件系统是为了取代 ISO-9660 文件格式而制定的，最终作为官方 DVD 视频/音频文件系统得到采用。

7-Zip 处理 UDF 文件主要是使用 CInArchive::ReadFileItem 方法。由于卷可能具有不止一个分区映射，所以卷对象会保存在一个对象容器中。在开始寻找某个项时，CInArchive::ReadFileItem 方法会尝试使用分区映射的对象容器以及长分配描述符的“PartitionRef”字段，来引用正确的对象。由于此方法不会检查“PartitionRef”字段是否大于可用的分区映射对象的数量，所以会导致读取越界，在一些情况下，还可能造成任意代码执行漏洞。

存在漏洞的代码：

CPP\7zip\Archive\Udf\UdfIn.cpp

```
Line 898    FOR_VECTOR (fsIndex, vol.FileSets)

Line 899    {

Line 900      CFileSet &fs = vol.FileSets[fsIndex];
```

```
Line 901    unsigned fileIndex = Files.Size();

Line 902    Files.AddNew();

Line 903    RINOK(ReadFileItem(volIndex, fsIndex, fs.RootDirICB, kNumRecursionLevelsMax));

Line 904    RINOK(FillRefs(fs, fileIndex, -1, kNumRecursionLevelsMax));

Line 905  }

(...)

Line 384    HRESULT CInArchive::ReadFileItem(int volIndex, int fsIndex, const CLongAllocDesc &lad, int numRecurseAllowed)

Line 385    {

Line 386      if (Files.Size() % 100 == 0)

Line 387        RINOK(_progress->SetCompleted(Files.Size(), _processedProgressBytes));

Line 388      if (numRecurseAllowed-- == 0)

Line 389        return S_FALSE;

Line 390      CFile &file = Files.Back();

Line 391      const CLogVol &vol = LogVols[volIndex];

Line 392      CPartition &partition = Partitions[vol.PartitionMaps[lad.Location.PartitionRef].PartitionIndex];
```

如果输入任何包含错误长分配描述符的代码，就会触发此漏洞。可以看出，在上面列出的代码的第 898-905 行，程序会搜索特定卷中的元素，文件集以 RootDirICB 长分配描述符起始。

此记录可以被故意设置错误，以达到恶意目的。漏洞出现在第 392 行，此处的 PartitionRef 字段会超过 PartitionMaps 容器中的元素的数量。

TALOS-CAN-0093，堆溢出漏洞，[CVE-2016-2334]

7-Zip 的 Archive::NHfs::CHandler::ExtractZlibFile 方法中，存在一个可被利用的堆溢出漏洞。在 HFS+ 文件系统中，文件可以使用 zlib 压缩格式进行存储。根据数据的大小，可以使用 3 种不同的方法来保存该格式的数据。压缩后大小超过 3800 字节的文件中的数据会存储在一个资源分支 (Resource Fork) 中，并被分为多个块。

块大小信息及其偏移值会保存在资源分支数据头后的一个表中。在执行解压缩之前，ExtractZlibFile 方法会从文件中读取块大小及其偏移值。然后，该方法会将读取的块数据缓存到静态大小缓冲区“buf”。但是，该方法不会检查块大小是否大于缓冲区“buf”的大小，这就导致有可能存在超过缓冲区“buf”大小的错误块大小，从而造成缓冲区溢出，并引发堆损坏。

存在漏洞的代码：

7zip\src\7z1505-src\CPP\7zip\Archive\HfsHandler.cpp

```
Line 1496    HRESULT CHandler::ExtractZlibFile(
Line 1497        ISequentialOutStream *outStream,
Line 1498        const CItem &item,
Line 1499        NCompress::NZlib::CDecoder *_zlibDecoderSpec,
Line 1500        CByteBuffer &buf,
Line 1501        UInt64 progressStart,
Line 1502        IArchiveExtractCallback *extractCallback)
```

```
Line 1503       {
Line 1504         CMyComPtr inStream;
Line 1505         const CFork &fork = item.ResourceFork;
Line 1506         RINOK(GetForkStream(fork, &inStream));
Line 1507         const unsigned kHeaderSize = 0x100 + 8;
Line 1508         RINOK(ReadStream_FALSE(inStream, buf, kHeaderSize));
Line 1509         UInt32 dataPos = Get32(buf);
Line 1510         UInt32 mapPos = Get32(buf + 4);
Line 1511         UInt32 dataSize = Get32(buf + 8);
Line 1512         UInt32 mapSize = Get32(buf + 12);
(...)
Line 1538         RINOK(ReadStream_FALSE(inStream, tableBuf, tableSize));
Line 1539
Line 1540         UInt32 prev = 4 + tableSize;
Line 1541
Line 1542         UInt32 i;
```

```
Line 1543        for (i = 0; i < numBlocks; i++)
Line 1544        {
Line 1545          UInt32 offset = GetUi32(tableBuf + i * 8);
Line 1546          UInt32 size = GetUi32(tableBuf + i * 8 + 4);
Line 1547          if (size == 0)
Line 1548            return S_FALSE;
Line 1549          if (prev != offset)
Line 1550            return S_FALSE;
Line 1551          if (offset > dataSize2 ||
Line 1552              size > dataSize2 - offset)
Line 1553            return S_FALSE;
Line 1554          prev = offset + size;
Line 1555        }
Line 1556
Line 1557        if (prev != dataSize2)
Line 1558          return S_FALSE;
```

```
Line 1559
Line 1560       CBufInStream *bufInStreamSpec = new CBufInStream;
Line 1561       CMyComPtr bufInStream = bufInStreamSpec;
Line 1562
Line 1563       UInt64 outPos = 0;
Line 1564       for (i = 0; i < numBlocks; i++)
Line 1565       {
Line 1566         UInt64 rem = item.UnpackSize - outPos;
Line 1567         if (rem == 0)
Line 1568           return S_FALSE;
Line 1569         UInt32 blockSize = kCompressionBlockSize;
Line 1570         if (rem < kCompressionBlockSize)
Line 1571           blockSize = (UInt32)rem;
Line 1572
Line 1573         UInt32 size = GetUi32(tableBuf + i * 8 + 4);
Line 1574
```

```
Line 1575          RINOK(ReadStream_FALSE(inStream, buf, size)); // !!!HEAP OVERFLOW !!!
```

我们设计了上面的代码，来模拟这样一种情况：从 HFS+ 映像中提取数据，其中包含几个具有“com.apple.decmpfs”属性的压缩文件，而且数据存储在一个资源分支中。

压缩文件数据被分为多个块，每个块在解压缩之前会被读取到缓冲区“buf”中（如第 1575 行所示）。根据“大小”值 ReadStream_FALSE（其实实际上就是 ReadFile API），一部分数据会被读取到缓冲区“buf”。

缓冲区“buf”的定义及大小，可以从 ExtractZlibFile 的调用函数（即 CHandler::Extract 方法）中找到。可以看到，“buf”的大小是一个常量，等于 0x10010 字节。

```
Line 1633      STDMETHODIMP CHandler::Extract(const UInt32 *indices, UInt32 numItems,
Line 1634          Int32 testMode, IArchiveExtractCallback *extractCallback)
Line 1635      {
(...)
Line 1652
Line 1653        const size_t kBufSize = kCompressionBlockSize; // 0x10000
Line 1654        CByteBuffer buf(kBufSize + 0x10); // we need 1 additional bytes for uncompressed chunk header
(...)
Line 1729        HRESULT hres = ExtractZlibFile(realOutStream, item, _zlibDecoderSpec, buf,
```

```
Line 1730        currentTotalSize, extractCallback);
```

再来看 ExtractZlibFile 方法，第 1573 行设置了从 tableBuf 读取的块的“大小”值。而在第 1538 行可以看到，tableBuf 是从文件读取的，这意味着此“大小”仅仅是从文件本身获得的一部分数据，所以我们可以影响此处的值。将“大小”值设置为大于 0x10010 就会造成缓冲区溢出，并导致堆损坏。

在第 1573 行之前，变量“大小”的值便已在第 1543-1555 行的一个循环中读取。这部分代码用于检查数据块是否一致，具体规则包括：

- 数据块应紧随在 tableBuf 之后开始（第 1540 行）
- 后续数据块应在前一个数据块大小加上偏移值的位置处开始（第 1549 行）
- 偏移值不应大于 dataSize2（压缩数据的大小）（第 1551 行）
- “大小”不应大于剩余的数据（第 1552 行）

正如我们所见，这部分代码没有检查“大小”是否大于“buf”大小的规则。而上述限制规则也没有对此产生影响。

总结

遗憾地说，不能正确验证输入数据的应用都会存在很多安全漏洞。像本文介绍的两种 7-Zip 漏洞，都是因为应用存在输入验证缺陷而导致的。因为数据来源有可能不可信任，所以对所有应用而言，数据输入验证对于确保应用的安全性至关重要。Talos 通过与 7-Zip 开发者协作，以负责任的态度披露并修复这些漏洞。建议用户尽快将存在漏洞的 7-Zip 版本升级到最新版本（版本 16.00）。

TALOS-CAN-0093 可通过 Snort 规则 38323-38326 进行检测。
TALOS-CAN-0094 可通过 Snort 规则 38293-38296 进行检测。

发布者：JAESON SCHULTZ；发布时间：上午 11:30

标签：7-ZIP、漏洞