2016 年 6 月 21 日，星期二

漏洞聚焦：PIDGIN 漏洞

漏洞发现者：Yves Younan。

Pidgin 是全球数百万系统上使用的通用聊天客户端。通过 Pidgin 聊天客户端，用户可以在多个聊天网络上同时进行通信交流。Talos 已确定，Pidgin 处理 MXit 协议的方式中存在多个漏洞。这些漏洞归为以下四类：

- 信息泄漏
- 拒绝服务
- 目录遍历
- 缓冲区溢出

已确定的漏洞列示如下（按 CVE 编号列出）：

CVE-2016-2365 - Pidgin MXIT 标记命令拒绝服务漏洞
CVE-2016-2366 - Pidgin MXIT 表命令拒绝服务漏洞
CVE-2016-2367 - Pidgin MXIT 头像长度内存泄露漏洞
CVE-2016-2368 - Pidgin MXIT g_snprintf 多缓冲区溢出漏洞
CVE-2016-2369 - Pidgin MXIT CP SOCK REC TERM 拒绝服务漏洞
CVE-2016-2370 - Pidgin MXIT 自定义资源拒绝服务漏洞
CVE-2016-2371 - Pidgin MXIT 扩展配置文件代码执行漏洞
CVE-2016-2372 - Pidgin MXIT 文件传输长度内存泄露漏洞
CVE-2016-2373 - Pidgin MXIT 联系人情绪拒绝服务漏洞
CVE-2016-2374 - Pidgin MXIT MultiMX 消息代码执行漏洞
CVE-2016-2375 - Pidgin MXIT 建议联系人内存泄露漏洞
CVE-2016-2376 - Pidgin MXIT 读取阶段 Ox3 代码执行漏洞
CVE-2016-2377 - Pidgin MXIT HTTP 内容长度缓冲区溢出漏洞
CVE-2016-2378 - Pidgin MXIT get_utf8_string 代码执行漏洞
CVE-2016-2380 - Pidgin MXIT mxit_convert_markup_tx 信息泄漏漏洞
CVE-2016-4323 - Pidgin MXIT 启动图像任意文件覆盖漏洞

信息泄漏漏洞

Talos 已发现，在实施作为 Pidgin 聊天客户端一部分的 MXit 协议时，存在四种信息泄漏漏洞。这些漏洞包括：

- CVE-2016-2367
- CVE-2016-2372
- CVE-2016-2375
- CVE-2016-2380

CVE-2016-2375

利用此漏洞，从服务器发送的经特殊设计的 MXIT 数据可能导致越界读取。在 mxit/protocol.c 文件第 2020 行的 mxit_parse_cmd_suggestcontacts 函数中，属性数量将从传入的数据包读取到变量计数中。

```
2020 count = atoi( records[0]->fields[3]->data );
```

接下来，此值用作第 2030 行循环的界限，循环索引用作第 2034-2036 行的数组索引。

```
2030 for ( j = 0; j < count; j++ ) {

2034 fname = records[0]->fields[4 + j]->data; /* field name */
2035 if ( records[i]->fcount > ( 2 + j ) )
2036   fvalue = records[i]->fields[2 + j]->data; /* field value */
```

随后，在这些位置设置的指针将可用于读取数据，可能会导致越界读取并将数据复制到结果字段，例如在第 2056-2059 行会出现这种情况：

```
2056 else if ( strcmp( CP_PROFILE_FULLNAME, fname ) == 0 ) {
2057 /* nickname */
2058   g_strlcpy( profile->nickname, fvalue, sizeof( profile->nickname ) );
2059   }
```

如果由于大多数信息不发送回服务器而使内存页面无法访问，则大多数越界读取只会导致崩溃。但是，当按下添加按钮时，则会调用 mxit/profile.c 中定义的回调。这会导致当昵称返回到服务器时，信息泄漏回服务器，因为它包括从内存中泄漏的信息。

CVE-2016-2380

在 Pidgin 中处理 MXIT 协议时存在信息泄漏。发送到服务器的经特殊设计的 MXIT 数据可能导致越界读取。可以确信用户输入了一个特定字符串，然后，该字符串错误地进行转换，导致可能的越界读取。

当 Pidgin 向服务器发送消息时，必须将标记从 libpurple（基于 HTML）标记转换为 MXIT 标记。要执行此操作，就需要调用 markup.c 文件中定义的 mxit_convert_markup_tx 函数。此函数将旧字符串消息中的数据复制到新的字符串 mx，同时转换此函数。

但是，在第 1146-1154 行，它会转换标记以更改字体颜色，而不检查剩余字符串的长度。

```
1146 else if ( purple_str_has_prefix( &message[i], "<font color=" ) ) {
1147 /* font colour */
1148 tag = g_new0( struct tag, 1 );
1149 tag->type = MXIT_TAG_COLOR;
1150 tagstack = g_list_append( tagstack, tag );
1151 memset( color, 0x00, sizeof( color ) );
1152 memcpy( color, &message[i + 13], 7 );
1153 g_string_append( mx, color );
1154 }
```

它将比较第 1146 行的消息在当前位置中的字符串是否以 <font color= 开头。如果是以此开头，则将 1 个元素中的 7 个字节复制到“=”末尾之后，可能跳过“#”标记。但是，如果“<font color=”位于字符串结尾，则会导致消息的越界读取。由于将跳过“=”末尾之后一个字节，将跳过 NULL 终止字符串，使该字符串后的 7 字节数据复制到 mx，这就是将要发送到服务器的字符串。

除了 DoS 问题外，CVE-2016-2367 和 CVE-2016-2372 还可能导致信息泄漏。CVE-2016-2367 可能导致敏感信息从内存泄露到头像之后的数据，然后，复制头像时可以转换头像。CVE-2016-2372 可能将内存中的敏感信息附加到已接收的文件的结尾，从而泄漏敏感信息。

拒绝服务漏洞

在实施作为 Pidgin 聊天客户端一部分的 MXit 协议时，Talos 已发现六种 DoS 漏洞。

- CVE-2016-2365
- CVE-2016-2366
- CVE-2016-2367
- CVE-2016-2369
- CVE-2016-2370
- CVE-2016-2372

CVE-2016-2365

在此漏洞中，通过服务器发送的经特殊设计的 MXIT 数据可能导致空指针取消引用。发生这种情况是因为处理不足的检查来验证是否已提供所有必填字段，以便成功执行收到的命令。当在消息中收到命令时，调用 mxit_parse_command() 函数，查找“键=值”格式的值，并将这些对插入散列表中。

```
580 hash = command_tokenize(start); /* break into <key,value> pairs */
```

然后，将检查正在处理何种命令，并调用正确的函数。

以下两种函数

command_imagestrip()
command_table()

依赖于如果未定义就会导致空指针取消引用的键/值对。第一个函数 command_imagestrip() 在 mxit/formcmds.c 中的第 383 行定义。在第 393-399 行，它将查找键 nm、v 和 dat 的值：

```
392 /* image strip name */
393 name = g_hash_table_lookup(hash, "nm");
394
395 /* validator */
396 validator = g_hash_table_lookup(hash, "v");
```

```
397
398 /* image data */
399 tmp = g_hash_table_lookup(hash, "dat");
```

虽然第 400 行中有一项检查可确保 tmp 不为 NULL，但对于名称和验证程序，没有类似的检查。这会导致当在第 419 和 420 中使用名称和验证程序时，发生空指针取消引用。

```
419 escname = g_strdup(purple_escape_filename(name));
420 escvalidator = g_strdup(purple_escape_filename(validator));
```

键 fw、fh 和 layer 在第 432-439 行具有类似的错误：

```
432 tmp = g_hash_table_lookup(hash, "fw");
433 width = atoi(tmp);
434
435 tmp = g_hash_table_lookup(hash, "fh");
436 height = atoi(tmp);
437
438 tmp = g_hash_table_lookup(hash, "layer");
439 layer = atoi(tmp);
```

在 mxit/formcmds.c 第 530-543 行定义的 command_table() 函数中，也会出现类似错误：

```
530 tmp = g_hash_table_lookup(hash, "col");
531 nr_columns = atoi(tmp);
532
533 /* number of rows */
534 tmp = g_hash_table_lookup(hash, "row");
535 nr_rows = atoi(tmp);
536
537 /* mode */
538 tmp = g_hash_table_lookup(hash, "mode");
539 mode = atoi(tmp);
540
541 /* table data */
542 tmp = g_hash_table_lookup(hash, "d");
543 coldata = g_strsplit(tmp, "~", 0);
```

如果任何这些键/值对缺失，就会发生崩溃。恶意服务器或拦截网络流量的攻击者可能会发送无效数据，触发此漏洞并导致崩溃

CVE-2016-2366

在此漏洞中，通过服务器发送的经特殊设计的 MXIT 数据可能导致越界读取。在 mxit/formcmds.c 第 531 和 535 行的 command_table 函数中，表的行数和列数是从服务器接收的。

```
531 nr_columns = atoi(tmp);
```

```
535 nr_rows = atoi(tmp);
```

然后，这两个值用在第 547 和 548 行的循环中，以访问第 549 行中的数组。

```
547 for (i = 0; i < nr_rows; i++) {
548 for (j = 0; j < nr_columns; j++) {
549 purple_debug_info(MXIT_PLUGIN_ID, " Row %i Column %i = %s\n", i, j, coldata[i*nr_columns + j]);
550    }
551 }
```

从数组读取的数据不会传送到服务器，因此没有信息泄漏，但是如果访问的内存内容位于无法访问的内存页上，仍会导致崩溃。恶意服务器或拦截网络流量的攻击者可能会发送无效数据，触发此漏洞并导致崩溃。

CVE-2016-2367

利用此漏洞，通过服务器发送的经特殊设计的 MXIT 数据可能导致越界读取。当通过 MXIT 服务器接收头像时，服务器将发送 CP_CHUNK_GET_AVATAR 命令。它将由 mxit/protocol.c 第 2208-2234 行上的 mxit_parse_cmd_media 函数进行处理。在第 2215 行，

```
2215 mxit_chunk_parse_get_avatar( &records[0]->fields[0]->data[sizeof( char ) +
sizeof( int )], records[0]->fields[0]->len, &chunk );
```

它将调用 mxit_chunk_parse_get_avatar() 函数，该函数将从 mxit/chunk.c 第 683 行的数据读取数据块大小。

```
683 pos += get_int32( &chunkdata[pos], &(avatar->length) );
```

如果指定的数据块长度大于缓冲区，会导致越界读取，内存中的结果数据将写在收到的头像后面。取决于触发漏洞时程序的内存布局，这会导致由于页面无法访问而使程序崩溃的情况，或导致内存中的敏感数据泄漏到文件的情况。用户可以选择是将此头像复制到其他位置，还是将其发送给其他用户，这会导致此数据泄漏。

恶意服务器、拦截网络流量的攻击者或恶意用户可能发送无效的文件大小，从而触发越界读取漏洞。这可能造成拒绝服务，或者，如果头像发送给另一个用户或同一用户，则可能导致信息泄漏。

CVE-2016-2369

利用此漏洞，通过服务器发送的经特殊设计的 MXIT 数据可能导致 NULL 指针取消引用。当从 MXIT 服务器接收数据以解析流入 MXIT 数据包的相关字节流时，可调用 mxit/protocol.c 中的 mxit_parse_packet() 函数。

当收到数据包时，在数据包中创建一条新记录以反映此数据：

```
2672 rec = NULL;
2673 field = NULL;
2674 memset( &packet, 0x00, sizeof( struct rx_packet ) );
2675 rec = add_record( &packet );
```

add_record 函数执行以下操作：

```
2634 static struct record* add_record( struct rx_packet* p )
2635     {
2636 struct record* rec;
2637 rec = g_new0( struct record, 1 );
2638 p->records = g_realloc( p->records,
2639       sizeof( struct record* ) * ( p->rcount + 1 ) );
2640 p->records[p->rcount] = rec;
2641 p->rcount++;
2642 return rec;
2643     }
```

这将在数据包中创建一条记录，并将 rcount 变量递增 1。

在第 2679-2744 行，进一步分析数据包，根据使用的分隔符是 0x0、0x1 还是 0x2，将数据包拆分成记录和字段。

如果特别感兴趣，请注意以下代码：

```
2679 while ( ( i < session->rx_i ) && ( !pbreak ) ) {
2680     switch ( session->rx_dbuf[i] ) {
2681 case CP_SOCK_REC_TERM :
2682 /* new record */
2683   if ( packet.rcount == 1 ) {
2684 /* packet command */
2885
2686 packet.cmd = atoi( packet.records[0]->fields[0]->data );
2687  }
```

CP_SOCK_REC_TERM 值表示到达记录的末尾，将会检索数据包发送的命令。但是，如果数据包以 NULL 字节开头，则该记录的 fields 变量尚未初始化，当尝试在第 2686 行取消引用该变量时将导致崩溃。通过服务器发送的经特殊设计的 MXIT 数据可能导致拒绝服务漏洞。恶意服务器可能发送以 NULL 字节开头的数据包，从而触发漏洞。

CVE-2016-2370

利用此漏洞，通过服务器发送的经特殊设计的 MXIT 数据可能导致越界读取。mxit/chunk.c 文件中定义的 mxit_chunk_parse_cr() 函数用于解析自定义资源，例如新的启动图像。这些资源类型作为多媒体数据包的一部分发送。

在第 573 行，从正在解析的数据块读取数据块长度，没有进行界限检查。此数据块还包含第 577 行中设置的一个或多个资源数据块。资源数据块的大小包含在该数据块的顶部，在第 587 行和第 604 行再次读取其大小，且没有进行界限检查：

```
573 pos += get_int32( &chunkdata[pos], &chunklen );
574
575 /* parse the resource chunks */
576 while ( chunklen > 0 ) {
577 gchar* chunk = &chunkdata[pos];
```

```
578
579 /* start of chunk data */
580 pos += MXIT_CHUNK_HEADER_SIZE;
581
582 switch ( chunk_type( chunk ) ) {
583 case CP_CHUNK_SPLASH : /* splash image */
584 {
585 struct splash_chunk* splash = g_new0( struct splash_chunk, 1 );
586
587 mxit_chunk_parse_splash( &chunkdata[pos], chunk_length( chunk ), splash );
588
589 cr->resources = g_list_append( cr->resources, splash );
590 break;
591 }
592 case CP_CHUNK_CLICK : /* splash click */
593 {
594 struct splash_click_chunk* click = g_new0( struct splash_click_chunk, 1 );
595
596 cr->resources = g_list_append( cr->resources, click );
597 break;
598 }
599 default:
600 purple_debug_info( MXIT_PLUGIN_ID, "Unsupported custom resource chunk received (%i)\n",
chunk_type( chunk) );
601 }
602
603 /* skip over data to next resource chunk */
604 pos += chunk_length( chunk );
605 chunklen -= ( MXIT_CHUNK_HEADER_SIZE + chunk_length( chunk ) );
```

此长度随后在第 582 和 587 行中用于访问数据块中的数据，从而导致越界读取。此数据不发送回服务器，因此不可能导致信息泄漏漏洞，但是，如果访问的位置不是分配的内存区域，则访问超出界限的内存时会导致拒绝服务。

CVE-2016-2372

利用此漏洞，通过服务器发送的经特殊设计的 MXIT 数据可能导致越界读取。当通过 MXIT 服务器接收文件传输时，服务器将发送 CP_CHUNK_GET 命令。它将由 mxit/protocol.c 第 2195-2206 行上的 mxit_parse_cmd_media 函数进行处理。

```
2195 case CP_CHUNK_GET : /* get file response */
2196 {
2197 struct getfile_chunk chunk;
2198
2199 /* decode the chunked data */
2200 memset( &chunk, 0, sizeof( struct getfile_chunk ) );
2201 mxit_chunk_parse_get( &records[0]->fields[0]->data[sizeof( char ) + sizeof( int )], records[0]-
>fields[0]->len, &chunk );
2202
2203 /* process the getfile */
2204 mxit_xfer_rx_file( session, chunk.fileid, chunk.data, chunk.length );
2205 }
2206 break;
```

在第 2201 行中，它会调用 mxit_chunk_parse_get 函数，该函数将文件大小（和文件中的其他一些信息）读取到 getfile_chunk 结构中。此函数是在文件 mxit/chunk.c 中定义的，其长度在第 509 行进行读取：

```
509 pos += get_int32( &chunkdata[pos], &(getfile->length) );
```

此函数解析信息并返回后，将在第 2204 行调用 mxit_xfer_rx_file 函数。

此函数是在 mxit/filexfer.c 文件中定义的，在第 445 行，它将收到的缓冲区中的信息读取到文件中：

```
445 if ( fwrite( data, datalen, 1, xfer->dest_fp ) > 0 ) {
```

如果指定的数据块长度大于缓冲区，会导致越界读取，内存中的结果数据将写在新收到的文件后面。取决于触发漏洞时程序的内存布局，这会导致由于页面无法访问而使程序崩溃的情况，或导致内存中的敏感数据泄漏到文件的情况。因为文件可能是无害的，受害者会将其发送给其他人甚至发送回攻击者，导致此数据泄露。

目录遍历漏洞

Talos 已发现，在实施作为 Pidgin 聊天客户端一部分的 MXit 协议时，存在一种目录遍历漏洞。此漏洞为

- CVE-2016-4323

CVE-2016-4323

在 Pidgin 中处理 MXIT 协议时存在目录遍历。从服务器发送的经特殊设计的 MXIT 数据可能导致覆盖文件。恶意服务器或具有网络流量访问权的人员可以为启动图像提供无效文件名，从而触发漏洞。

Pidgin 允许 MXIT 服务器提供连接到服务器时要显示的启动图像。服务器还可以借助从服务器返回的命令，通过提供新的图像更新此图像。

当服务器通过多媒体命令提供新的图像时，将在 mxit/protocol.c 的第 2170 行调用 splash_update 函数：

```
2170 splash_update( session, chunk.id, splash->data, splash->datalen, clickable );
```

在第 564 行的 mxit_chunk_parse_cr 函数中，从来自服务器的数据读取 chunk.id 变量：

```
564 pos += get_utf8_string( &chunkdata[pos], cr->id, sizeof( cr->id ) );
```

splash_update 函数是在第 115-136 行的 mxit/splashscreen.c 中定义的：

```
115 void splash_update(struct MXitSession* session, const char* splashId, const char* data, int datalen,
gboolean clickable)
116 {
117 char* dir;
118 char* filename;
119
```

```
120 /* Remove the current splash-screen */
121 splash_remove(session);
122
123 /* Save the new splash image */
124 dir = g_strdup_printf("%s" G_DIR_SEPARATOR_S "mxit", purple_user_dir());
125      purple_build_dir(dir, S_IRUSR | S_IWUSR | S_IXUSR);
126       /* ensure directory exists */
127 filename = g_strdup_printf("%s" G_DIR_SEPARATOR_S "%s.png", dir,
purple_escape_filename(splashId));
128      if (purple_util_write_data_to_file_absolute(filename, data, datalen)) {
129 /* Store new splash-screen ID to settings */
130 purple_account_set_string(session->acc, MXIT_CONFIG_SPLASHID, splashId);
131
132 purple_account_set_bool(session->acc, MXIT_CONFIG_SPLASHCLICK, clickable );
133 }
134 g_free(dir);
135 g_free(filename);
136         }
```

在第 127 行中，正确转义 splashId，以防止发生目录遍历。但是，未转义的字符串存储在第 130 行的 MXIT_CONFIG_SPLASHID 变量中。在此函数第 121 行中调用的 splash_remove 函数将使用 MXIT_CONFIG_SPLASHID 查找要删除的文件（第 84-104 行）：

```
 84 void splash_remove(struct MXitSession* session)
 85 {
 86 const char* splashId = NULL;
 87 char* filename;
 88
 89 /* Get current splash ID */
 90 splashId = splash_current(session);
 91
 92 if (splashId != NULL) {
 93 purple_debug_info(MXIT_PLUGIN_ID, "Removing splashId: '%s'\n", splashId);
 94
 95 /* Delete stored splash image */
 96 filename = g_strdup_printf("%s" G_DIR_SEPARATOR_S "mxit" G_DIR_SEPARATOR_S "%s.png",
purple_user_dir(), splashId);
```

```
 97 g_unlink(filename);
 98 g_free(filename);
 99
100 /* Clear current splash ID from settings */
101 purple_account_set_string(session->acc, MXIT_CONFIG_SPLASHID, "");
102 purple_account_set_bool(session->acc, MXIT_CONFIG_SPLASHCLICK, FALSE);
103 }
104 }
```

但是，与 splash_update 不同，此例中没有文件名转义，这样会使攻击者在系统上删除任意 png 文件。

缓冲区溢出漏洞

Talos 已发现，在实施作为 Pidgin 聊天客户端一部分的 MXit 协议时，存在五种目录遍历漏洞。这些漏洞包括：

- CVE-2016-2368
- CVE-2016-2371
- CVE-2016-2376
- CVE-2016-2377
- CVE-2016-2378

CVE-2016-2368

利用此漏洞，通过服务器发送的经特殊设计的 MXIT 数据可能导致缓冲区溢出。Pidgin 的 MXIT 插件在大约 27 个接收 g_snprintf 函数返回值的位置使用此函数。当 g_snprintf 返回时，它将返回缓冲区足够大时写入的字节数，而不是实际写入的字节数量。在

https://developer.gnome.org/glib/stable/glib-String-Utility-Functions.html#g-snprintf 中介绍这方面的相关信息：

MXIT 插件使用 g_snprintf 返回值作为在多个位置进行处理的字符串的索引或偏移量，但不确保返回值在界限内。这可能导致缓冲区溢出。在 mxit_queue_packet() 函数的第 467-479 行阐述了此问题的示例。

```
467 hlen = g_snprintf( header, sizeof( header ), "id=%s%c", purple_account_get_username( session->acc ), CP_REC_TERM ); /* client mxitid */
468
469 if ( session->http ) {
470 /* http connection only */
471 hlen += g_snprintf( header + hlen, sizeof( header ) - hlen, "s=");
472 if ( session->http_sesid > 0 ) {
473 hlen += g_snprintf( header + hlen, sizeof( header ) - hlen, "%u%c", session->http_sesid, CP_FLD_TERM ); /* http session id */
474 }
475 session->http_seqno++;
476 hlen += g_snprintf( header + hlen, sizeof( header ) - hlen, "%u%c", session->http_seqno, CP_REC_TERM ); /* http request sequence id */
477 }
478
479 hlen += g_snprintf( header + hlen, sizeof( header ) - hlen, "cm=%i%c", cmd, CP_REC_TERM ); /* packet command */
```

第 467 行返回的长用户帐户可能导致第 471、473、476 或 479 行缓冲区溢出。尽管建议检查 g_snprintf 的所有返回值，但顾问（提供链接）特别指出遍布 7 个函数的 12 个调用可能是问题最多的，因为它们会将可能来自不可信位置的数据复制到字符串中。恶意服务器或潜在的恶意用户（如果数据未经过服务器验证）会发送将与 g_snprintf 一起错误使用的数据，从而可能导致缓冲区溢出。

CVE-2016-2371

利用此漏洞，通过服务器发送的经特殊设计的 MXIT 数据可能导致缓冲区溢出。当从服务器接收扩展配置文件数据包时，调用 mxit_parse_cmd_extprofile() 函数。在第 1837 行，它将读取服务器发送到变量计数的属性数。

```
1837 count = atoi( records[0]->fields[1]->data );
```

接下来，此值用作第 1839 行循环的界限，用于计算第 1843 行数组的索引，此值随后用于访问第 1845-1847 行的数组中的值。

```
1839 for ( i = 0; i < count; i++ ) {
1840 char* fname;
1841 char* fvalue;
1842 char* fstatus;
1843 int f = ( i * 3 ) + 2;
1844
1845 fname = records[0]->fields[f]->data; /* field name */
1846 fvalue = records[0]->fields[f + 1]->data; /* field value */
1847 fstatus = records[0]->fields[f + 2]->data;/* field status */
```

索引还用于写入第 1859-1860 行的数组，这可能导致超出界限的写入。

```
1859 fvalue[10] = '\0';
1860   records[0]->fields[f + 1]->len = 10;
```

恶意服务器、拦截网络流量的攻击者或潜在恶意用户（如果数据未经过服务器验证）可能发送无效的记录数，从而导致超出界限的数据写入。

CVE-2016-2376

从服务器发送的经特殊设计的 MXIT 数据可能导致缓冲区溢出。mxit/protocol.c 文件中的 mxit_cb_rx 函数是回调函数，每当从 MXIT 服务器发送数据时，都由 Pidgin 调用。当接收数据时，还会在第 2825 行接收传入的数据包的大小。第 2826 行有一项检查，可确保此数据不大于定义为 CP_MAX_PACKET 的 MXIT 数据包的最大大小。

```
2825 session->rx_res = atoi( &session->rx_lbuf[3] );
2826 if ( session->rx_res > CP_MAX_PACKET ) {
purple_connection_error( session->con, _( "A connection error occurred to MXit.(read stage 0x03)" ) );
2827 }
```

这也是数据读入的缓冲区的大小。不过，如果该大小大于 CP_MAX_PACKET，则会记录一个错误，但执行仍会继续。而且，如果大小为负数（这是可能的，因为 rx_res 属于 int 类型），则不会记录错误，执行仍会继续。此大小随后用在第 2846 行的读取操作中。

```
2846 len = read( session->fd, &session->rx_dbuf[session->rx_i], session->rx_res );
```

恶意服务器或拦截网络流量的攻击者可能发送无效的数据包大小，从而触发缓冲区溢出。

CVE-2016-2377

由服务器发送的经特殊设计的 MXIT 数据可能导致一个字节的超出界限写入。当接收来自 HTTP 服务器的对 HTTP 请求的回应时，将调用 mxit/http.c 中定义的 mxit_cb_http_read() 回调函数。此函数将解析 HTTP 信头，然后送出正文，以作为常规 MXIT 数据包进行处理。在发生 HTTP 信头解析的过程中，从第 178-185 行的信头读取 CONTENT_LENGTH：

```
178 ch += strlen( HTTP_CONTENT_LEN );
179 tmp = strchr( ch, '\r' );
180 if ( !tmp ) {
181                 purple_debug_error( MXIT_PLUGIN_ID, "Received bad HTTP reply
                                        packet (ignoring packet)\n" );
182 goto done;
183 }
184 tmp = g_strndup( ch, tmp - ch );
185 bodylen = atoi( tmp );
```

bodylen 定义为有符号的整数，因此，从 HTTP 信头读取的输入可能是负数。在第 189-192 行有一项大小检查：

```
189 if ( buflen + bodylen >= CP_MAX_PACKET ) {
190 /* this packet is way to big */
191 goto done;
192 }
```

但是，如果 bodylen 设置为负值，将通过此检查。

在第 206 行，bodylen 复制到作为无符号整数的变量 session->rx_i，从而将潜在的负数 bodylen 转换为大的正值。

```
206 session->rx_i = bodylen;
```

此值随后用于在第 2669 行的 mxit/procotol.c 中的 mxit_parse_packet 函数中处理数据包时控制循环：

```
2669 while ( i < session->rx_i ) {
```

该索引随后使用多个位置写入缓冲区 rx_dbuf，包括第 2713、2720 和 2729 行。这会允许攻击者在缓冲区 rx_dbuf 执行缓冲区溢出。

恶意服务器会发送负的 content-length，以响应 HTTP 请求，从而触发漏洞。

CVE-2016-2378

在处理 Pidgin 中的 MXIT 协议时存在缓冲区溢出漏洞。通过服务器发送的经特殊设计的 MXIT 数据可能导致缓冲区溢出，从而造成内存崩溃。libpurple/protocols/mxit/chunk.c 第 231 行中定义的 get_utf8_string 函数将最大字符串长度作为参数。通常，它会作为写入的字符串 str 的大小而传入。

它将读取第 238 行的字符串长度，检查并确保它不大于第 240 行的最大字符串长度。如果是这样，则会将长度设置为等于 maxstrlen。

```
238 pos += get_int16( &chunkdata[pos], &len );
239
240 if ( len > maxstrlen ) {
```

```
243   skip = len - maxstrlen;
244 len = maxstrlen;
245 }
```

不过，len 是将要从 nthos 读取的有符号短整型，nthos 将会读取无符号整型，但是由于 len 是有符号的，因此它将转换为有符号整数。如果 len 值是一个大的正值，它将转换为一个负值，绕过第 240 行的大小检查。

第 248 行对 get_data 的调用会导致缓冲区溢出：

```
248 pos += get_data( &chunkdata[pos], str, len );
```

get_data 函数将结束调用预期为无符号大小参数的 memcpy，并将负值解释为大的正值。

恶意服务器或潜在恶意用户（如果数据未经过服务器验证）会发送负长度值，从而触发此漏洞。

总结

使用即时消息软件快速高效地通信，使此软件非常受欢迎。因此，攻击者不断尝试查找并利用即时消息应用中的漏洞，因为这可以使他们访问大量潜在的受害者。修补软件对于减少这些不断变化的持续攻击的攻击面非常重要。许多用户不能做到定期修补，这会使攻击者轻易获得系统的访问权限。Pidgin 即时消息客户端安装在许多系统上，安装最新更新版本 (2.11.0) 对于防止 Talos 所确定的漏洞至关重要。可以在此处找到最新的 pidgin 软件版本。

缓解措施

为了保护我们的客户，Talos 已经发布了相关规则来检测利用这些漏洞的攻击尝试。请注意，Talos 未来可能会发布更多规则，当前规则会根据未来得到的更多漏洞信息而有所变更。如需获取有关最新规则的信息，请参阅防御中心、FireSIGHT 管理中心或 Snort.org。

Snort 规则：38344、38345、38545-38551、38578、38867、38870、39150、39151

发布者：EARL CARTER；发布时间： 15:37

标签：缓冲区溢出、PIDGIN、漏洞