LiveZilla实时聊天应用7大漏洞解析

Pinging / 2019-08-11 09:11:00 / 浏览数 3627 安全技术 漏洞分析 顶(0) 踩(0)

在2019年6月，Fortinet的FortiGuard实验室发现了LiveZilla的Live Chat中的7个漏洞，并进行了相关报告撰写。LiveZilla是一家拥有众多财富500强企业和顶尖大学用户的软件公司，拥有超过15,000名用户。

我们可以在8.0.1.0及更低版本中找到这些漏洞。 在撰写此博客时，这些问题已得到修复，并且供应商已发布了这些漏洞的修复程序。 FortiGuard Labs对供应商的快速响应和及时修复表示赞赏。

以下是漏洞的摘要：

1. LiveZilla Server before 8.0.1.1 is vulnerable to SQL Injection in server.php via the p_ext_rse parameter
2. LiveZilla Server before 8.0.1.1 is vulnerable to XSS in mobile/index.php via the Accept-Language HTTP header
3. LiveZilla Server before 8.0.1.1 is vulnerable to Denial of Service (memory consumption) in knowledgebase.php via a large integer value of the depth parameter
4. LiveZilla Server before 8.0.1.2 is vulnerable to XSS in the chat.php Create Ticket Action
5. LiveZilla Server before 8.0.1.1 is vulnerable to SQL Injection in functions.internal.build.inc.php via the parameter p_dt_s_d.
6. LiveZilla Server before 8.0.1.1 is vulnerable to XSS in the ticket.php Subject
7. LiveZilla Server before 8.0.1.1 is vulnerable to CSV Injection in the Export Function

漏洞细节

1. FG-VD-19-082 8.0.1.1之前的LiveZilla Server通过p_ext_rse参数进行server.php的SQL注入攻击

在审核`\livezilla\server.php`中的源代码文件时，第76行处表示server.php将导入intern.php文件。

```
73
74          header("Connection: close");
75          header("Cache-Control: no-cache, must-revalidate");
76      ----------require(LIVEZILLA_PATH . "intern.php");
77
```


当我们查看`\livezilla\intern.php`时，我们看到它在第29行调用类`OperatorRequest`的`Listen()`方法。该类派生自`\livezilla\_lib\objects.internal.in`

```
26  if($_POST[POST_INTERN_SERVER_ACTION]==INTERN_ACTION_LISTEN || $_POST[POST_INTERN_SERVER_ACTION]==INTERN_ACTION_LOGIN)
27  {
28      Server::$Operators[CALLER_SYSTEM_ID]->SaveMobileParameters();
29      OperatorRequest::Listen();
30      if(STATS_ACTIVE && !LOGIN)
31          Server::$Statistic->ProcessAction(ST_ACTION_LOG_STATUS,array(Server::$Operators[CALLER_SYSTEM_ID]));
32  }
```


正如我们在图中看到的，它然后在第302行调用同一类中的`Build()`方法：

```
290     static function Listen()
291     {
292         OperatorRequest::Process();
293
294         if(!Server::IsServerSetup() && !LOGIN && Server::$Operators[CALLER_SYSTEM_ID]-
295             return;
296
297         Server::$Response->XML = "<listen disabled=\"".base64_encode(((Server::IsAvail
                ])) ? "ex_time=\"<!--execution_time-->\"" : "").">\r\n";
298         Server::$Response->Typing = "";
299         if(Server::$Response->Login != null)
300             Server::$Response->XML .= Server::$Response->Login;
301
302     ........OperatorRequest::Build();
303
304         processPosts();
```

深入研究Build()，第405行处我们可以看到它调用了buildResources()：

```
386     static function Build()
387     {
388         require_once(LIVEZILLA_PATH . "_lib/functions.internal.build.inc.php");
389         Server::$Operators[CALLER_SYSTEM_ID]->GetExternalObjects();
390 
391         buildIntern();
392         demandVisitors();
393         demandVisitorBrowsers();
394         demandVisitorBrowserURLs();
395         demandObjectData();
396         demandChats();
397         demandEvents();
398         buildActions();
399 
400         if(!Server::IsServerSetup())
401         {
402             if(!LOGIN)
403             {
404                 buildNewPosts();
405 ····················buildResources();
406                 demandFeedback();
407                 demandFilters();
408                 demandTickets();
409                 demandEmails();
410                 demandChatArchive();
411                 demandReports();
412             }
413         }
414     }
```

```
51  function buildResources($xml = "", $last = 0)
52  {
53      if(isset($_POST[POST_INTERN_XMLCLIP_RESOURCES_END_TIME]))
54      {
55          if($_POST[POST_INTERN_XMLCLIP_RESOURCES_END_TIME] == XML_CLIP_NULL)
56              $_POST[POST_INTERN_XMLCLIP_RESOURCES_END_TIME] = 0;
57
58          $count = 0;
59          if($result = DBManager::Execute(true, "SELECT * FROM `" . DB_PREFIX .
                DATABASE_RESOURCES . "` WHERE `edited` >'" . DBManager::RealEscape($
                _POST[POST_INTERN_XMLCLIP_RESOURCES_END_TIME]) . "' AND
                `discarded`<2 AND `edited`<=" . DBManager::RealEscape(time()) . "
                AND `parentid` NOT IN (100,101,102) ORDER BY `edited` ASC"))
60          {
61              while($row = DBManager::FetchArray($result))
62              {
63                  $res = new KnowledgeBaseEntry($row);
64                  if(++$count <= DATA_ITEM_LOADS || $res->Edited == $last)
65                      $xml .= $res->GetXML();
66                  else
67                      break;
68                  $last = $res->Edited;
69              }
70          }
71      }
72      Server::$Response->Resources = (strlen($xml) > 0) ? $xml : null;
73  }
```

正如在图5中的第59行所见，它执行以下SQL查询：

```
SELECT * FROM `" . DB_PREFIX . DATABASE_RESOURCES . "` WHERE `edited` > " .
DBManager::RealEscape($_POST[POST_INTERN_XMLCLIP_RESOURCES_END_TIME]) . " AND
`discarded`<2 AND `edited`<=" . DBManager::RealEscape(time()) . " AND `parentid` NOT IN
(100,101,102) ORDER BY `edited` ASC
```

在列表中，使用`DBManager::RealEscape`过滤器函数清理参数以避免SQL注入。 然而，这里缺少对引用清理，这使得过滤功能变得无效。
因此，我们只需要在SQL查询中输入没有任何引号的值，以便利用此漏洞。

`$_POST[POST_INTERN_XMLCLIP_RESOURCES_END_TIME])`被定义在`\livezilla\_definitions\definitions.protocol.inc.php`中：

```
50  define("POST_INTERN_XMLCLIP_CHATS_CLOSED_END_TIME","p_ext_clc");
51  define("POST_INTERN_XMLCLIP_RESOURCES_END_TIME","p_ext_rse");
52  define("POST_INTERN_XMLCLIP_REPORTS_END_TIME","p_st_r");
53  define("POST_INTERN_XMLCLIP_TICKETS_END_TIME","p_ext_me");
```

因此，该漏洞的最终payload如下所示：

p_ext_rse=(select*from(select(if((substr(123,1,1) like 1),2,sleep(5))))a)

p_ext_rse=(select*from(select(if((substr(123,1,1) like 2),2,sleep(5))))a)

图7显示了供应商提供的补丁情况：

```
57
58
59  `edited` > '" . DBManager::RealEscape($_POST[POST_INTERN_XMLCLIP_RESOURCES_END_TIME]) . "' AND `discarded`<2 A
60
61
```

2. LiveZilla Server通过Accept-Language HTTP头受到mobile/index.php中的XSS攻击

在分析第84行的\livezilla\mobile\index.php中的源代码文件时，我们发现服务器在没有进行清理的情况下回应了$language，这可能导致跨站点脚本（XSS）漏洞

```
82      <script type="text/javascript">
83          var translationData = <?php echo $jsLanguageData; ?>;
84          var detectedLanguage = <?php echo "'".$language."'"; ?>;
85          var logit = function(myString) {
```


$language的值取自$ _SERVER ['HTTP_ACCEPT_LANGUAGE']，这是HTTP请求标头中的Accept-Language字段。

```
11  $language = $_SERVER['HTTP_ACCEPT_LANGUAGE'];
12  $language = explode(',', $language);
13  $language = strtolower($language[0]);
```


通过使用Man-in-The-Middle■MiTM■攻击方法，或者任何扩展来修改标头，攻击者可以在用户的浏览器中运行javascript代码。

图10显示了供应商提供的补丁：

```
27  if(!isset($LANGUAGES[strtoupper($language)]))
28  {
29      /*$parts = explode("-",$language);
30      if(strlen($language) <= 6 && !(is_array($parts) && count($parts)
31          exit("Invalid localization data " . $language);
32      */
33      $language = "en";
34  }
```


3.FG-VD-19-084 LiveZilla Server很容易受到knowledgebase.php中的拒绝服务的影响

这个拒绝服务被发现在\livezilla\knowledgebase.php，第39到51行：

```
39  if(!empty(Server::$Configuration->File["gl_kbmr"]))
40  {
41      if(!isset($_GET["depth"]))
42          $_GET["depth"] = -1;
43
44      $path = "./";
45
46      for($i=0;$i<$_GET["depth"];$i++)
47          $path .= "../";
48
49      $html = str_replace("<!--path-->",$path,$html);
50      $html = str_replace("<!--searchtarget-->","knowledge-base/",$html);
51  }
```

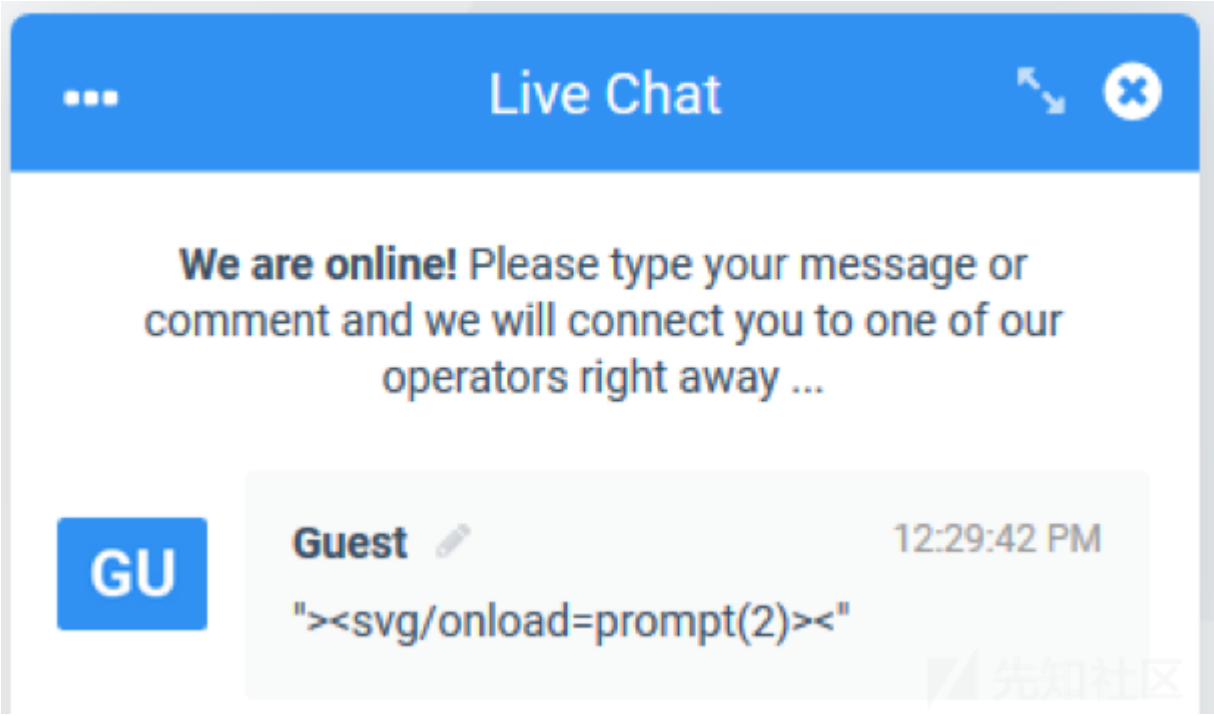

第39行的条件结构确定是否打开了搜索引擎优URL选项。 如果是，则查找GET参数深度，然后对其值执行基于循环的操作，该操作可由攻击者控制。
换句话说，如果我们提供输入，比如“■depth = 2200000”，它将循环2200000次。
正如我们在图11中的第46-47行所示，循环指令将字符串“../”连接到$path变量中，这可能导致内存溢出。

图12显示了供应商提供的补丁：

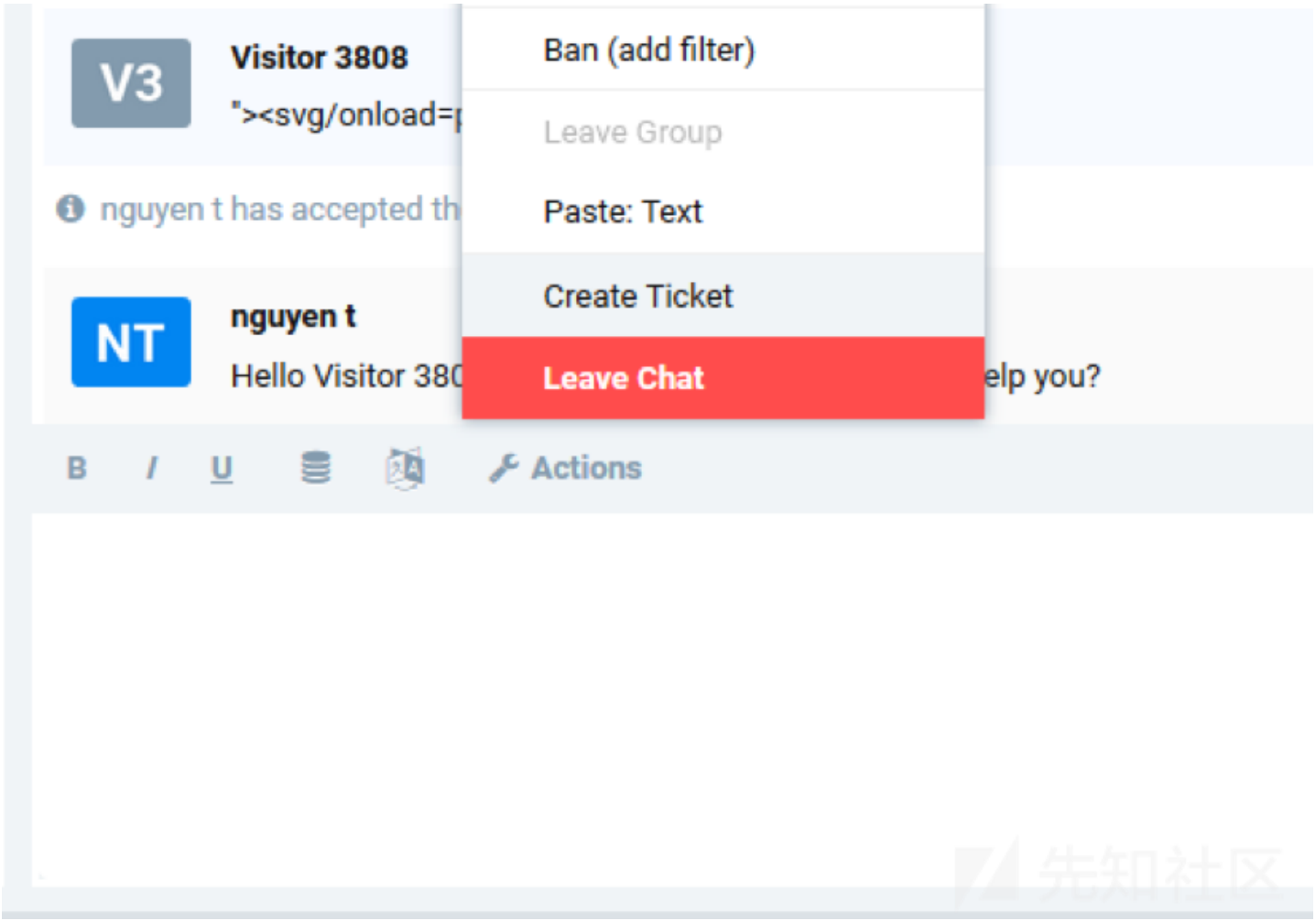
```
39  if(!empty(Server::$Configuration->File["gl_kbmr"]))
40  {
41      if(!isset($_GET["depth"]))
42          $_GET["depth"] = -1;
43
44      $path = "./";
45
46      for($i=0;$i<min(25,$_GET["depth"]);$i++)
47          $path .= "../";
48
49      $html = str_replace("<!--path-->",$path,$html);
50      $html = str_replace("<!--searchtarget-->","knowledge-base/",$html);
51  }
```

4. LiveZilla Server在chat.php创建故障单操作中容易受到XSS攻击

这是另一个可以从Guest Live Chat窗口触发的XSS漏洞。 攻击者可以在实时聊天中输入XSS的payload。



在管理面板中，如果管理员在聊天窗口中创建了一个故障单，则该聊天内容将呈现为一个新的聊天记录弹出窗口而不进行清理，这可能导致在用户的浏览器中执行任意javas



在验证供应商的补丁后，我们意识到版本8.0.1.1中的补丁不完整。
我们通知了开发人员，并为他们提供了可以绕过8.0.1.1版补丁的额外payload，他们为此问题提供了完整的解决方案。 图15显示了供应商版本8.0.1.2中的补丁：

```
686 eValue(myCustomInput,customValue[0].text);«¶
687 ·'-';«¶
688 ;«¶
689
690
691
692
693
694 lass)·+·'">'·+·lzm_commonTools.HighlightSearchKey(lzm_commonTools.htmlEntities(inputText),CommonInput
695
```

5. LiveZilla Server很容易通过参数p_dt_s_d在functions.internal.build.inc.php中进行SQL注入

另一个SQL注入漏洞可以在 \livezilla_lib\functions.internal.build.inc.php,中的第596到605行找到。

```
596         $q_sort = array();
597         $q_sort["id"] = " GROUP BY `".DB_PREFIX.DATABASE_TICKETS."`.`id` ORDER BY `".DB_PREFIX.
                DATABASE_TICKETS."`.`id` " . (!empty($_POST["p_dt_s_d"]) ? $_POST["p_dt_s_d"] : "DESC");
598         $q_sort["update"] = " GROUP BY `".DB_PREFIX.DATABASE_TICKETS."`.`id` ORDER BY `".DB_PREFIX.
                DATABASE_TICKETS."`.`last_update` " . (!empty($_POST["p_dt_s_d"]) ? $_POST["p_dt_s_d"] : "DESC");
599         $q_sort["priority"] = " GROUP BY `".DB_PREFIX.DATABASE_TICKETS."`.`id` ORDER BY `".DB_PREFIX.
                DATABASE_TICKETS."`.`priority` " . (!empty($_POST["p_dt_s_d"]) ? $_POST["p_dt_s_d"] : "DESC") . "
                ,`".DB_PREFIX.DATABASE_TICKETS."`.`last_update` DESC";
600         $q_sort["wait"] = " GROUP BY `".DB_PREFIX.DATABASE_TICKETS."`.`id` ORDER BY `".DB_PREFIX.
                DATABASE_TICKETS."`.`wait_begin` " . (!empty($_POST["p_dt_s_d"]) ? $_POST["p_dt_s_d"] : "ASC") . "
                ,`".DB_PREFIX.DATABASE_TICKETS."`.`last_update` DESC";
601         $q_sort["status"] = " GROUP BY `".DB_PREFIX.DATABASE_TICKETS."`.`id` ORDER BY `te`.`status` " . ((!
                empty($_POST["p_dt_s_d"]) ? $_POST["p_dt_s_d"] : "ASC") . ",`te`.`sub_status` ASC,`".DB_PREFIX.
                DATABASE_TICKETS."`.`last_update` DESC");
602         $q_sort["sub_status"] = " GROUP BY `".DB_PREFIX.DATABASE_TICKETS."`.`id` ORDER BY
                if(`te`.`sub_status` = '' or `te`.`sub_status` is null,1,0),`te`.`sub_status` " . (!empty($_POST["
                p_dt_s_d"]) ? $_POST["p_dt_s_d"] : "ASC");
603         $q_sort["channel_type"] = " GROUP BY `".DB_PREFIX.DATABASE_TICKETS."`.`id` ORDER BY
                field(`channel_type`,4,1,6,5,3,2,7,0) " . ((!empty($_POST["p_dt_s_d"]) ? $_POST["p_dt_s_d"] : "ASC
                ") . ",`sub_channel` ASC");
604         $q_sort["sub_channel"] = " GROUP BY `".DB_PREFIX.DATABASE_TICKETS."`.`id` ORDER BY if(`sub_channel` =
                '' or `sub_channel` is null,1,0),`sub_channel` " . (!empty($_POST["p_dt_s_d"]) ? $_POST["p_dt_s_d"
                ] : "ASC");
605
```


服务器确定参数p_dt_s_d是否通过POST HTTP请求发送，然后将其值直接输入到查询而不清除该值。 这导致了经典的SQL注入。

图17和18显示了供应商提供的补丁：

```
596 $q_sort = array();
597
598 $sortdefdesc = !empty($_POST["p_dt_s_d"]) && $_POST["p_dt_s_d"] == "ASC" ? "ASC" : "DESC";
599 $sortdefasc = !empty($_POST["p_dt_s_d"]) && $_POST["p_dt_s_d"] == "DESC" ? "DESC" : "ASC";
```

6. LiveZilla Server易受ticket.php中的XSS攻击

在第109行的`\livezilla\ticket.php`中发现了另一个XSS。对于此漏洞，服务器使用我们制作的内容替换了`$subject`持有者，而没有进行适当的清理。

```
106             $html = str_replace("<!--close_button_style-->",$status == 0 || $status == 2 ||
107             $html = str_replace("<!--id-->",$_GET["id"],$html);
108             $html = str_replace("<!--tid-->",Encoding::Base64UrlEncode($_GET["id"]),$html);
109             $html = str_replace("<!--subject-->",$subject,$html);
110             $html = str_replace("<!--tickets-->",$thtml,$html);
111             $html = str_replace("<!--status-->",$status,$html);
112             $html = str_replace("<!--salt-->",$ticket->Salt,$html);
```

图21显示了供应商提供的补丁：

```
106  $html = str_replace("<!--close_button_style-->",$status == 0 || $status == 2 || $status == 3
107  $html = str_replace("<!--id-->",$_GET["id"],$html);
108  $html = str_replace("<!--tid-->",Encoding::Base64UrlEncode($_GET["id"]),$html);
109  $html = str_replace("<!--subject-->",htmlentities($subject,ENT_QUOTES,"UTF-8"),$html);
110  $html = str_replace("<!--tickets-->",$thtml,$html);
111  $html = str_replace("<!--status-->",$status,$html);
```

7. LiveZilla Server易受导出功能中的CSV注入攻击

我们还在源代码文件`\livezilla\_lib\functions.internal.man.inc.php`中发现了逗号分隔值（CSV）文件注入。
从图22中的第736行到第744行，我们可以看到服务器尝试以CSV格式导出数据而不进行清理。

```
736                 if($_POST["p_export_format"] == "CSV")
737                 {
738                     $fp = fopen($filePath, "w");
739                     while($row = @DBManager::FetchArray($result,"ASSOC"))
740                     {
741                         fputcsv($fp, $row);
742                     }
743                     fclose($fp);
744                 }
```

图23显示了供应商提供的补丁：

```
736                 if($_POST["p_export_format"] == "CSV")
737                 {
738                     $fp = fopen($filePath, "w");
739                     while($row = @DBManager::FetchArray($result,"ASSOC"))
740                     {
741                         foreach($row as $i => $data)
742                         {
743                             if(in_array(substr($data,0,1),array("+","-","=","@")))
744                                 $row[$i] = "'" . $data;
745                         }
746 
747                         fputcsv($fp, $row);
748                     }
749                     fclose($fp);
750                 }
```

漏洞发现时间点

2019年6月22日：Fortinet向LiveZilla报告了FG-VD-19-082和FG-VD-19-084的漏洞
2019年6月24日：Fortinet报告了漏洞FG-VD-19-086
2019年6月25日：Fortinet向LiveZilla报告了FG-VD-19-083，FG-VD-19-085，FG-VD-19-087和FG-VD-19-088的漏洞
2019年6月26日：LiveZilla确认了漏洞，发布了针对这些漏洞的补丁
2019年6月27日：Fortinet确认了这些漏洞的修复，但FG-VD-19-085除外
2019年7月1日：LiveZilla确认FG-VD-19-085的修复不正确，等待8.0.1.2版本
2019年7月23日：LiveZilla发布了8.0.1.2补丁漏洞，Fortinet确认修复了FG-VD-19-085

结论

总之，所有这些漏洞的根本原因是缺乏简单的输入清理。 因此，FortiGuard Labs在LiveZilla Live Chat软件中发现了多个漏洞，范围从中等严重级到严重级。

Live Chat用户立即应用LiveZilla提供的修补程序至关重要，因为某些漏洞（例如启用SQL注入的漏洞）将允许攻击者在成功利用后从数据库中提取机密信息。

■■■■■■■■■■■https://www.fortinet.com/blog/threat-research/livezilla-live-chat-technical-advisory.html

上一篇：De1CTF2019 官方Writ... 下一篇：开源ids ，snort and ...
1. 0 条回复
   - 动动手指，沙发就是你的了！