# CVE-2019-6976：Libvips 内存泄露

Hulk / 2019-04-25 08:49:00 / 浏览数 4190 安全技术 漏洞分析 顶(0) 踩(0)

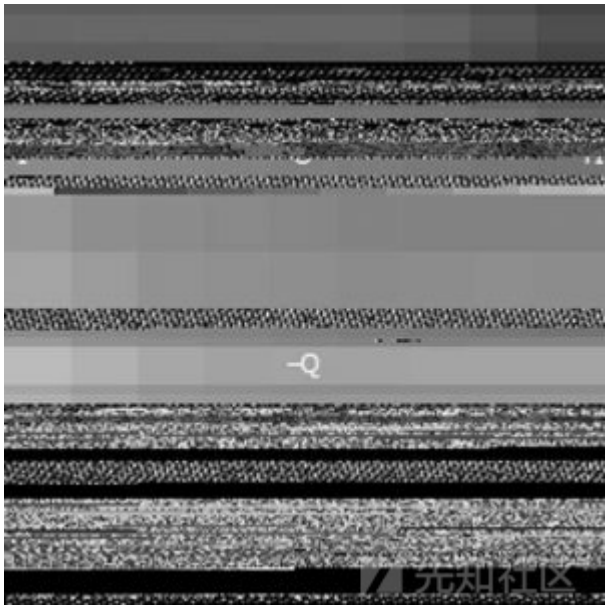
---

文章来源：https://blog.silentsignal.eu/2019/04/18/drop-by-drop-bleeding-through-libvips/

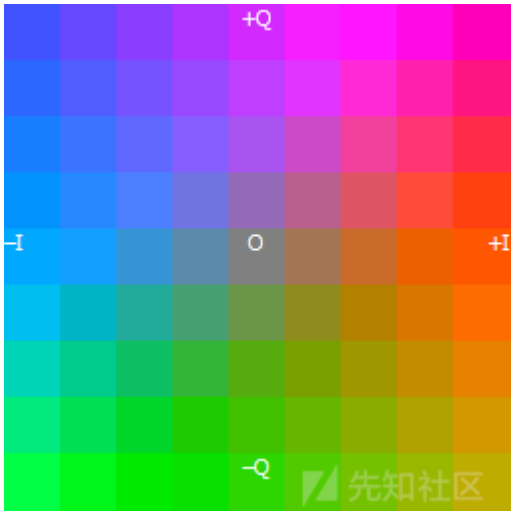
## 前言

在最近的一次渗透测试中，目标网站上有一个常见的功能：上传个人图片。使用Burp-Suite的UploadScanner扩展扫描上传点，但没有有效结果。随后，我们注意到测试用
Evans挖掘到的一个雅虎漏洞，于是我们决定深入调查它。

我将这个可疑的图片下载到本地，它是这样的：




我们觉得它有点像"结构化噪音"（附有大面积空白和重复的图案），存在大面积相同的颜色以及一些显现的字符！为了弄清楚到底发生了什么，我们在Logger++（BP扩展）




第一张图片似乎是由这张图片衍生出来的。

## 影响预测

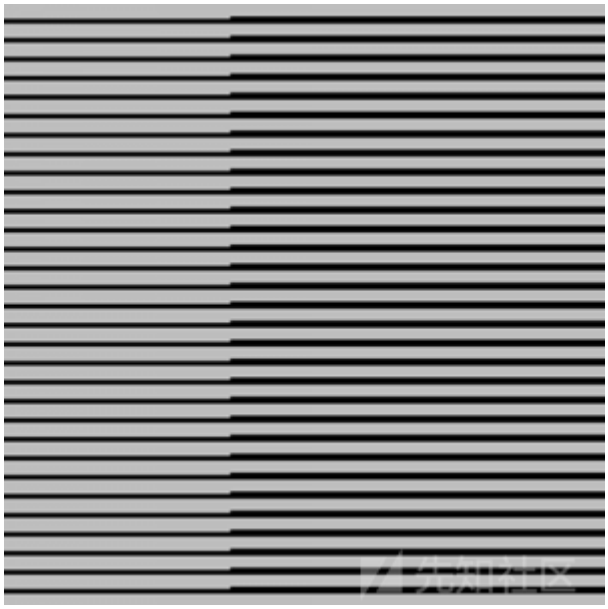
此时，我们仍然不知道这张"噪音"图片为什么以及是如何出现的：因为无法看到软件堆栈，不知道噪音是不是内存泄露或者只是从引起错误的图片生成的随机数据（缩略图）

- 这个Bug的确泄露了未初始化的内存，但我们看到的并不是压缩或者算法转化的结果。
- 程序是从不同用户会话之间共享的进程地址空间读取未初始化的内存，因此可以用户盗取用户会话。

## 漏洞溯源

目前我们仍然不知道是哪个软件造成这种现象。我们运行实际运行了fingerping，但是没有得到确切结果。有些幸运，目标程序上另一个接口泄露了该程序使用的库名称，该

我们把libvips
8.4.3下载到本地，编译后将fingerping的样本放入运行。结果与我们目标程序的反应完美相符。因为缩略图的地址空间没有包含太多使用过的数据，所以这里的未初始化内存

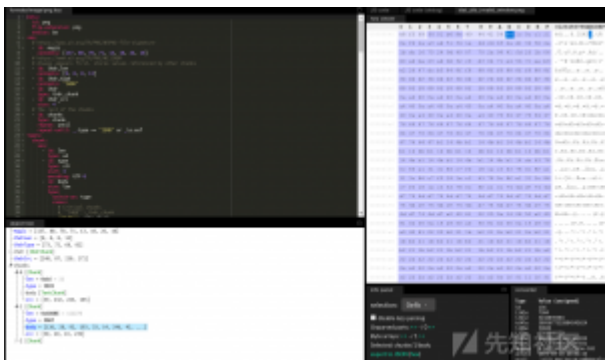


确认漏洞

目标程序使用libvips调整上传图像大小，这将导致单个像素和周围其他像素（挤压）变得“模糊”，使得输出解码不准确。只上传相同维度的文件或许可以解决这个问题。为了
windows”。在查阅RFC国际标准后，我们很快发现PNG使用的压缩方式为deflate，其中RFC_zlib声明了与deflate压缩有关的方式（CM=8）：

> 关于CM=8, CINFO是LZ77窗口大小的以2为底的对数，再减去8（CINFO=7等于一个32K大小的窗口）。在此版本的规范中不允许CINFO的值大于7。

由CM与CINFO字段组成的CMF字节再PNG文件中IDAT区块之后，开始经过delfate流。我们将图像使用Kaitai的在线IDE加载，可以清楚看到CM与CINFO都设置为8，这对

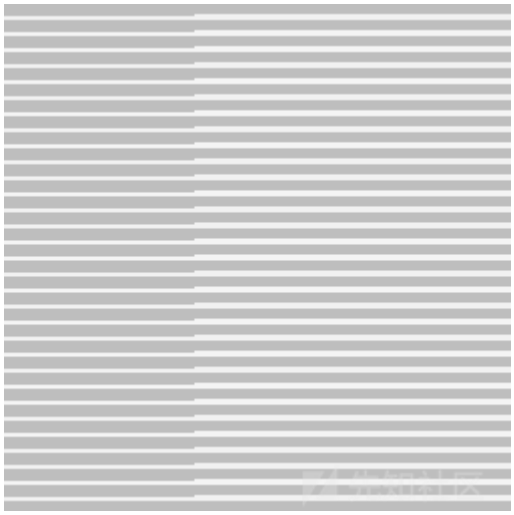


然后我们创建了一个尺寸恰当的测试图像，并且使用hex编辑器将CINFO区域编辑为8从而损坏窗口尺寸。同时，我们还修改了vipsthumbnail的源代码，使用已知的多个内存

```
void* allocs[10240];
int asizes[4] = {6088,95000,775,6088}; // Sizes determined after some hours of debugging...
for(int i = 0; i<10240; i++){
   allocs[i] = g_malloc(asizes[i % 4]);
   memset(allocs[i], 0xf1 + (i % 4), asizes[i % 4]);
}

for(int i=0;i<10240;i++){
   if ( 1|| i % 4 == 0 || i % 4 == 2 ){
       g_free(allocs[i]);
   }
}
```

使用新创建的图像进行测试，我们看到了一块漂亮的白色（ish）图案，附有预期`#f1f1f1-#f4f4f4`的值，这证明了可以造成位校正的内存泄露。



修复

在确认问题后，我们立即警告客户，并且提醒libvips和Sharp库的维护者。他们都在几个小时内回复，而且libvips的修复程序在当天就发布了。很幸运，修复程序非常简单，

```
@@ -173,7 +173,7 @@ vips_malloc( VipsObject *object, size_t size )
{
   void *buf;

-   buf = g_malloc( size );
+   buf = g_malloc0( size );

       if( object ) {
       g_signal_connect( object, "postclose",
@@ -317,7 +317,7 @@ vips_tracked_malloc( size_t size )
    */
   size += 16;

-       if( !(buf = g_try_malloc( size )) ) {
+       if( !(buf = g_try_malloc0( size )) ) {
#ifdef DEBUG
       g_assert_not_reached();
#endif /*DEBUG*/
```

然而，这并不是故事的结尾。因为我们在讨论一个开源库，为了使程序真正地修复，所有下游包和发行版都必须打上补丁。正如你在时间线看到的那样，这项工程花了十来
- 在写本文时Debian稳定版仍未修复。而且，我们的初衷是使Sharp开启[libvips深度防御措施]，当遇到引起错误的数据，将停止处理。

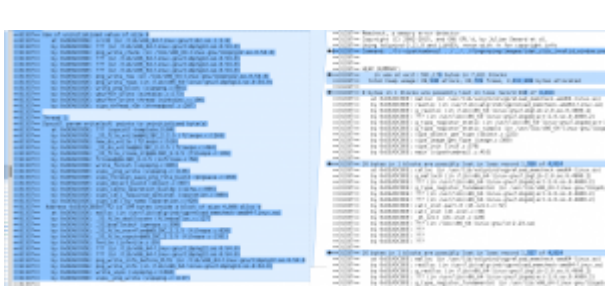
我们在MITRE上申请到CVE，编号为CVE-2019-6976。

进攻性建议

为了提高安全人员黑盒测试的速度与准确性，我们针对开源工具提出以下几点建议：

- 我们提议将libvips的指纹添加到fingerping库中。
- 我们在扩展UploadScanner上做了一些额外测试，该测试通过尝试压缩原始的点阵图数据来检测通过redownloader检索到的图像上的噪音。压缩率越低，则表明熵数据

对于白盒测试，在相关库中找出类似bug无疑是一个艰巨的任务：尽管libvips有自己的fuzzing设施，但未初始化的内存bug通常不会导致非法内存访问，因此这大概率不会



小结

本文我们详细描述了一次以前从未出现过的漏洞的发现以及识别的经历。就如我们前面提到的那样，由于检测这类bug非常困难，因此我们相信类似的漏洞潜伏在大量大大小

由于图像处理库任务的复杂性，它们易于出现有关不安全的内存处理漏洞。考虑到大规模图像处理任务，低级高性能的库似乎使最好的选择，但是大部分（web）应用并不需

- 首先，目前很难确认其他库上是否存在libvips库的组件。我们希望这些库在它们的文档中明确声明引用。
- 托管运行的服务器上的内置库也可能存在漏洞。尽管如此，它们仍然可以认为比从头开始编写的C/C++代码更安全。

从漏洞披露的角度来看，本次发现的漏洞的处理是理想的，受影响的每一方都能做出快速而专业的反应。同时，本次漏洞处理也印证了协同处理可以大大减轻修复工作量，但
Cupitt和Lovell Fuller的回应，技术参考以及协助修复！同时，我们还感谢所有参与及时向用户提供修复的人员！

时间线

2019-01-18：向libvips和Sharp报告Bug
2019-01-18：libvips 8.7.4发布，修复Bug
2019-01-18：通知Debian，RedHat和net-vips存在Bug
2019-01-18：Sharp更改为fail-fast机制
2019-01-18：在Homebrew中发现存在libvips组件。
2019-01-18：Debian更新
2019-01-18：NetVips更新
2019-01-18：Fedora 和Remi的RPM更新
2019-01-18：MacPorts更新
2019-01-19：在Alpine Linux aports中发现ibvips组件，并修复
2019-1-26：MITRE分配CVE-2019-6976，同时发布错误的漏洞信息
2019-3-31：Debian稳定版发布更新
2019-04-18：在blog.silentsignal.eu上发布博文
2019-04-18：要求MITRE更新CVE信息


上一篇：C++逆向学习(一) string 下一篇：Android反调试——从源码入手
1. 0 条回复
   - 动动手指，沙发就是你的了！