php代码隐写到图片,从PNG Dropper到Alpha通道隐写术实践

转载

蓝虫虫 于 2021-03-19 09:43:06 发布 58 收藏
文章标签： php代码隐写到图片

0x1 前言

前些天从安全客上看到一篇翻译文章是关于利用PNG像素隐藏PE代码，对实现细节很感兴趣就抽空稍微研究了下相关内容，于是就有了本次分享。

0x2 PNG Dropper样本分析

先来看下翻译文中的样本，其中PE资源里包含一个PNG目录，该目录里存储了8张PNG格式的图片：

将8张图片全部提取出来，随便选择一张打开查看，发现可以正常打开，但显示的内容都是一些杂乱的随机像素点：

将其拖进010edit分析下格式，能够正常解析图片数据，数据格式貌似没什么异常：

再来跟踪一下该样本内存解密PE的过程。整体流程是依次定位资源中的8张PNG图片，提取其中的图片数据到缓冲区里拼凑成完整的PE，最后再内存LoadPE执行。

然后最关心的部分应该就是提取图片数据的过程了。这部分的代码看上去比较多，其实主要也就3小部分。第1部分是将PNG图片流读进一个Bitmap对象来方便操作图片：

第2部分是获取该PNG的各项基本数据包括宽度、高度、Stride(行跨度？)和图像格式，这些数据主要是用作最后一步调用的结构体参数(Rect和BitmapData)，这部分最关键，因为这些参数直接影响最后获取到的图片数据。

最后1步其实很简单，调用“GdipBitmapLockBits”函数就能直接获取到图片的数据了。当然这里得到的图片数据不是上文010edit里看到的那个IDAT数据，而是对该数据根据第2步指定的图像格式参数来进行解码后的“原始像素”数据。

根据此流程也可以自己尝试用Gdi+写代码来提取图片数据，然后发现从上面样本中图片获取到的编码格式是“PixelFormat32bppARGB”，然而实际内部存储的像素数据却用的是“PixelFormat16bppARGB1555”编码格式，这可能是作者的一种反检测小技巧吧。

0x3 Alpha通道隐写实践

本来分析到上面也就结束了，结果看到安全客上有评论谈及隐写术，想起了大学时期使用Matlab做LSB图像信息隐藏的实验，当时只是简单应付课程对其懵懵懂懂实在有点遗憾，于是趁此机会重新复习下顺便研究看看所谓高级的隐写术——利用RGBA中的A(Alpha)通道来隐藏数据。

首先复习LSB(最低有效位)，这个是指图像隐藏采用的一种算法吧，查了一下这个算法看网上有用jpg图片来做，也有用bmp图片来做的，还有jpg处理完变png的，有点晕，其实应该是针对一般的RGB图像都能做。怎么理解呢，我们知道图像是由一个个的像素点构成，比如一张100×100(宽度和高度都是100)的图片，总共就有10000个像素点，而其中每个像素点都用一个RGB值来表示其颜色。对于不同的图片格式，其像素数据的编码格式可能不一样，RGB值的存储格式也就可能不同，即使相同格式比如PNG，其像素格式也有很多种，具体可参考“RGB格式详解”。本文主要以其中的“RGB32”或“ARGB32”为例，一个RGB值用3字节来分别表示R(红)、G(绿)、B(蓝)3个颜色通道的数值，这样每个颜色通道的数值范围都是0-255，LSB图像隐藏算法的原理就是利用其中每个像素颜色通道的最低位来存储需要隐藏的数据，这样对于整张图像的“视觉影响”非常低，图片处理前后就可能肉眼看上去“没什么变化”。

正好前段时间在看雪公众号上看到一篇用PHP来实现LSB的分享，直接贴上来先看看实现过程吧，最核心部分的就是将需要隐藏的数据的每个二进制位，依次存进每个选取像素点中蓝色(B)通道的最低位。

随便找一张图片测试一下效果吧：

完全看不出差异，只是隐藏的信息“hello”字符串实际上已经存储到右边图片的像素矩阵中，数据位依次被隐藏到所选45度角线上的像素点。当然这只是个简单的例子，我们自己写工具时可以从原点开始遍历所有像素点，也可以将其他2个通道最低位都用上以提高信息存储的容量。但总的来说，个人觉得LSB能够存储的容量还是相对比较小，比如我想把系统的计算器程序calc.exe(大约900kb)存进一张图片中则需要一张大约100万像素的图片。如下所示，经过自己实现的LSB工具(使用每个像素的最低位依次存储)处理后，将计算器程序放到图片中，图片的size前后膨胀了3MB！

信息提取的话只要反向操作像素数据就行，具体可以参见本文后面分享的Demo。这里就不继续探究LSB了，感兴趣的网上搜一下参考资料一大堆。接下来直接进入PNG的Alpha通道隐写这个话题，具体地本文是以PNG的“PixelFormat32bppARGB”这种像素格式的图像隐写为例进行分享。

其实我并不了解真正的Alpha通道隐写是什么样的，也没有去找相关实现的资料，此番分享完全是自己摸索一下基于“PixelFormat32bppARGB”格式的PNG图片如何进行信息隐藏。这种格式非常容易理解，就是在3字节RGB基础上再加一个A通道，也就是Alpha或者透明通道，于是正好组成4字节一组的像素数据(从高到低为ARGB)。然后对这个A通道的理解很关键，这个字节的数值表示的是该像素点的透明度：数值为0时，该像素点完全透明，相当于RGB完全隐藏不用；数值为255时该像素点只由RGB颜色决定，相当于没有透明度。如下图所示，左图的背景全透明，A通道数值为0,；右图背景设置(不)透明度为50%，则刚好为1字节数值范围的一半即0x80。

接着就很自然而然地想利用这种特性来藏数据了。既然透明度为0像素点完全透明，那该像素点的RGB这3个通道的字节数据我们不就可以随便用来存数据了么，而且其他像素点仍可以存储正常的图像数据啊！比如上图的透明背景图，透明像素数据的RGB值为0xffffff是因为该图的背景图层为白色(或空)，那么反正它也不显示(透明度100%)，干脆就征用一个字节来隐藏数据吧(如B通道字节，也可以RGB全征用)。那么隐藏数据前可以先计算一下图片的可用容量，比如设定A通道为0且R、G两通道为0xff则判断为可用像素点：

当可用透明像素点数量大于要隐藏的数据字节数就可以进行存储了，直接依次存进去每个可用像素点的B通道字节呗，存完后加个结束标志就行：

直接看一下隐藏效果吧，还是以存储计算器程序calc.exe(大约900kb)为例：

还行吧，肉眼仍看不出图片显示有什么不同，图片size看上去也还可以，膨胀不到1MB(差不多计算器程序的size)，并且还有不小的提升空间(比如图片处理掉更多像素，或者加入其他2个通道存储)。然后使用神器“Stegsolve.jar”来看看图片隐藏的计算器吧：

当然之所以能这样一开始直接就看到数据是因为选取图片像素的原点区域正好被掏空(透明区域)可以用来存储数据，其实根据载体图片的不同完全可能将字节数据“随机化”。最后关于如何获取具有透明通道格式的可存储图片也很简单，以Photoshop为例，随便拉张图片选取无关背景区域直接删除像素(抠图)，也可以再加一层透明图层背景，然后保存为PNG格式即可。

0x4 工具实现

既然研究都差不多了，不实现工具化总归有点不完整，示例代码就不多做介绍了，写的很粗糙原本是不打算献丑的，本着学习分享的理念还是直接扔到Github上仅供参考吧。以下分别是LSB和PNG Alpha通道隐写工具，均包含加解密两种功能，以及处理过程相关的提示，可以满足一般的信息隐藏需求。

包含上述直接编译好的工具可以测试，目前仅面向PNG的ARGB32格式图片支持良好~

文章封面图使用此方式将工具隐写其中，读者可以用此图实践操作。

0x5 总结

通过这次的探索，对图像有了进一步的理解，图像像素的组织格式多种多样，本质上是颜色信息存储格式的差异，而信息隐藏的原理却是异曲同工，都是在于对目标载体的数据格式做分析，寻求可供存储的“无关”数据空间，只要不影响原始载体的功能“感觉”，应该都是可以操作的。

0x6 参考链接