《电子证据分析》课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 实验名称 |  | 姓名 |  |
| 班级 |  | 学号 |  |
| 实验日期 |  | 指导老师 |  |
| 实验环境  (实验软硬件要求) | Win10、python、微信、cmd、volatility | | |
| 实验目的 | （给出本次实验所涉及并要求掌握的知识点）  *了解并且掌握volatility的相关基础插件的操作和使用*  *掌握内存取证的基本步骤和实际操作*  *能够自己掌握并且编写小程序解出微信的图片* | | |

|  |  |
| --- | --- |
| 实验内容 | 一、使用volatility对内存镜像进行操作。并且使用pslist、psscan、pstree、timeliner插件，并且描述对应的结果  先进入对应的路径，打出以下命令可以查看帮助    -f就是要输入的文件镜像    因为还需要知道文件对应的系统到底是什么，可以使用imageinfo这个插件    故，先使用的命令如下：  volatility\_2.6\_win64\_standalone.exe -f 内存镜像.dd imageinfo  这里要注意内存镜像一定要放在和volatility软件一个目录下才能这样，不知道为啥我用绝对路径会出错。    要注意imageinfo这个插件只有对windows系列系统才有效，对于其他的系统是无效的。    pslist插件:遍历EPROCESS结构体来输出所有运行的进程到屏幕上。（EPROCESS是一种双链表结构体，每有一个进程运行就会将其加入这个结构体中）PS.双链表结构体这里就是一个有分别指向着前、后两个结构体的指针  psscan插件:扫描所有的进程项。  对两者的结合使用：      上面的是对pslist里面的内容的解释，这里面的句柄数可以理解为是占用了的资源数。会话数（上面打错了……是会话）是与网络的交互，Exit是内核里这个进程结束的时间。      比较内容结果之后就可以发现了，差异就是有一个进程id为712的进程隐藏起来了    如果觉得这样的排列看着不方便找到进程之间的父子关系，还可以用pstree插件。它会把所有的没有被隐藏的进程都以树的方式来表示出来，看起来会更加直观。    这样就可以让这些内容显示的更加清楚了。  另外，还有一个插件tiimeliner也需要了解。  这个插件可以记录所有的进程的所有的操作。      如果内容过多，则可以使用管道命令重定向到对应的文件中去    这样就会在对应的路径下出现一个233.txt的文档，里面的内容就是之前在命令行里面出现的那些内容    二、编程，编写一个小程序能够将微信自己发送的图片文件按照原来的格式还原出来。  首先我们需要了解微信文件在哪里？  微信文件在C:\Users\<用户名>\Documents\WeChat Files\wxid\_<一堆字符> \FileStorage\Image 里面  并且里面是根据时间分成了多个文件夹的，具体如下：    打开文件夹可以看到都是一些.dat文件    其实，这些文件就是我们需要的图片了。只不过使用了一些特殊的办法进行了一些特殊的操作而已。因为我们可以看到，很多地方会出现连续相同的内容，很明显这个信息熵很低，没有加过密，如下图：    那么，微信到底用了什么操作呢？  其实就是用了一个特定的key，让它来和原本的数据异或，获得的新数据就是那个dat文件了……看起来很简单。  所以我的最初逻辑如下：  第一步，将dat文件的头两个字节先互相异或，然后再和其他的标准文件头异或出来的值进行比较，这样就能找出对应的文件类型。  第二步，将dat文件头第一个字节与已经确定的类型的文件的第一个字节进行异或，这样就相当于是密文（dat文件的第一个字节）与明文（已知的图片类型的第一个字节）异或来得到秘钥key  第三部，将秘钥key与dat文件内容进行不断的异或，并且将结果输出到一个新的文件中去。  这里首先就是异或判断文件类型的部分，这部分代码如下：    可以看到，我这里使用了很简单的变量存储已经确定的四种常见图片格式的异或结果，然后使用文件操作获取前两个字节，再将其异或（这里就是我的前面的逻辑的第一步），然后再将其与之前的存放好的变量进行比较，比较完成之后再将dat文件的第一个字节和确定的类型的第一个字节进行异或，得到秘钥key（这里就是我前面说的第二步），然后返回对应的秘钥key和对应的类型。（这里插一句，本来没有想着返回类型的字符串，结果发现不这样写没办法后面确定后缀……）  那么第三步就是全部的异或然后写入了。第三部的代码如下，其实也没有什么难点，就是异或，然后返回，这里我分成了三个函数，这样会小一些，也方便后面进一步的修改。（模块化总比一个main函数方便维护一些些吧……）    这里选择了直接在要解密的那个文件夹里面自己创建同名的图片文件，kind就是类型后缀  这里其实写入的时候还用了一个struct的库，将所有前面异或好了的数字值将其再打包成为byte类    这个函数就是很简单的异或操作，密文与秘钥异或得到原文，然后这个完整的原文作为一个返回值返回    这个函数就是前面做的一个简单的将byte类转换成为字符串类型这样就可以将前面二进制读取出来的内容直接变成可以进行转换操作的字符串了。例如：b”\xd3”→”\xd3”  综上就是整体的逻辑了  后面为了将其更加具有较强的功能性，我增加了能够支持递归解析子文件夹以及自动排除非dat文件（原理就是直接识别文件后缀，只要不是dat后缀的就直接跳过）。代码如下：    也其实是从上几次的作业里修改而来的，我只是改了改而已……  最后的调用：    下面就是完整的代码：  import binascii  import struct  import os  # 一个神奇的数学原理，所以，只要用原先压缩过的dat文件的值进行开头的异或，就能知道值了  # DA = A ^ K  # DB = B ^ K  # DA ^ DB = (A ^ K) ^ (B ^ K) = (A ^ B) ^ (K ^ K) = A ^ B  def IsJpgOrPng(dir):      PNG = 0x89 ^ 0x50      JPG = 0xff ^ 0xd8      TRFF = 0x49 ^ 0x49      BMP = 0x42 ^ 0x4d      with open(dir,"rb") as f1:          number = f1.read(2)          answer = int(number[0]) ^ int(number[1])          if (answer == PNG):              # print("%s is a PNG"%dir)              key = int(number[0]) ^ 0x89              kind = "png"          elif (answer == JPG):              # print("%s is a JPG"%dir)              key = int(number[0]) ^ 0xff              kind = "jpg"          elif (answer == TRFF):              key = int(number[0]) ^ 0x49              kind = "trff"          elif (answer == BMP):              key = int(number[0]) ^ 0x42              kind = "bmp"      return key,kind  def Xor(ciphertext,key):      answer = []      for i in ciphertext:          i = i ^ key          answer.append(i)      return answer  def BytToStr(string):      """此函数的作用就是将读取到的16进制字符流变成正常的字符列表，方便后面的异或"""      hexs = []      for s in string:          hexs.append(s)      return hexs  def change(file1):      f1 = open(r'%s'%file1,'rb')        length = os.path.getsize(r'%s'%file1)      # 动态获取要读取的文件的大小      key,kind = IsJpgOrPng(file1)      # 我的秘钥是28      byte = f1.read(length)      # 将读取到的内容先全部放入一个变量里面      ciphertext = BytToStr(byte)      answer = Xor(ciphertext,key)      filename = file1[:-4]        with open(r"%s.%s" % (filename, kind),"ab+") as fout:          for i in answer:              a = struct.pack('B',i)              fout.write(a)  def RecursiveFileSearch(path):      if (not os.path.exists(path)):  # 判断文件(文件夹)是否存在          print(path + "不存在！！")          return 0      if (os.path.isfile(path) and path[-3:] == 'dat') :          change(path)          print("文件%s已经修改完成"%path)      elif (os.path.isdir(path)):          dirlist = os.listdir(path)          for dir in dirlist:              dir = path + '\\' + dir              RecursiveFileSearch(dir)  if \_\_name\_\_ == "\_\_main\_\_":      path  = input("请输入文件夹或文件位置，支持自动递归解码，解码完成后的图片会保存在原来dat文件的位置")      RecursiveFileSearch(path) |
| 实验总结 | （对本次实验涉及到的知识点的重难点归纳、实验心得、思考与建议） |
| 评分 |  |