数据隐写到图片中

原创

深入浅出0 于 2019-05-07 10:03:55 发布 936 收藏
分类专栏： Web安全 文章标签： 隐写

Web安全 专栏收录该内容
25 篇文章 4 订阅
订阅专栏

背景

jpg图片文件开头标志为SOI，结束标志为EOI，直接将数据添加文件结尾即可（不影响图片的正常显示）

使用

- copy /b img.jpg + data.data result.jpg
- 或者cat data.data >> img.jpg 即可

python脚本

- png图片整个为分块结构,其中有四块是必须的
- PNG files start with an 8 byte signature, 89 50 4E 47 0D 0A 1A 0A. The first byte is a non-ASCII character, byte 2 through 4 spell out PNG in ASCII. The remaining bytes are line ends, the DOS EOF character, and another line break
- 接着是IHDR,IDAT,IEND三个必须块，这四块的顺序不能变；还有其他tEXt，bKGD，pHYs等非必须块
- 每个块结构包括size, type(IHDR etc.), data, CRC-check，其中crc校验的内容为type+data
- 。。。
- 增加任意类型的块 不会影响图片查看
- 增加或修改非必须块(tEXt等) 也不会影响图片查看
- 通过上面两个方式 可以实现隐藏code到png图片
- 。。。
- 下面示例代码可以隐藏数据到png文件,数据使用xtea加密

```python
#!/usr/bin/env python
# coding=utf-8

import binascii
import struct
from optparse import OptionParser
import hashlib

class Punk(object):

    _END_CHUNK_TYPE = 'IEND'
    _PUNK_CHUNK_TYPE = 'tEXt'
    _PUNK_PASS_TYPE = 'pAss'
    _MAX_BYTES = 2147483647
    _TEMP_CHUNK = ''
    _TEMP_BYTES = bytearray()
    _chunks = dict()

    def __init__(self):

        self._mode = None
        self._file = None
        self._output = None
        self._bytes_to_hide = None

        self._bytes_read = 0

    def decode(self, input_file, output_file):
        self._mode = 'decode'
        self._file = open(input_file, 'rb+')
        self._output = open(output_file, 'wb+')

        # First move cursor past the signature
        self._read_bytes(8)

        # Start reading chunks
        self._read_next_chunk()

    def encode(self, input_file, bytes_to_hide):

        self._mode = 'encode'
        self._file = open(input_file, 'rb+')
        self._bytes_to_hide = bytes_to_hide

        # First move cursor past the signature
        self._read_bytes(8)

        # Start reading chunks
        self._read_next_chunk()

    def _read_bytes_as_hex(self, position):

        return self._read_bytes(position).encode('hex')

    def _read_bytes_as_ascii(self, position):

        return self._read_bytes(position).encode('ascii')

    def _read_bytes_as_int(self, position):
        return int(self._read_bytes_as_hex(position), 16)

    def _read_bytes(self, byte_count):

        self._bytes_read += byte_count
        return self._file.read(byte_count)

    def _rewind_bytes(self, byte_count):

        self._bytes_read -= byte_count
        self._file.seek(self._bytes_read)

    def _inject_punk_chunk(self):

        # Move back 8 bytes.
        self._rewind_bytes(8)

        chunk_size = len(self._bytes_to_hide)
        #print 'Hiding', (chunk_size / 1024), 'kB (', chunk_size, 'bytes)'

        # Create a temporary byte array for the CRC check.
        tmp_bytes = bytearray()

        # First write the chunk type
        tmp_bytes.extend(bytearray(self._PUNK_CHUNK_TYPE))

        # Now write the bytes of whatever we're trying to hide
        tmp_bytes.extend(self._bytes_to_hide)

        #print 'Injecting punk chunk'

        # Write the chunk size
        self._file.write(bytearray(struct.pack('!i', chunk_size)))

        # And the type
        self._file.write(bytearray(self._PUNK_CHUNK_TYPE))

        self._file.write(self._bytes_to_hide)

        crc = binascii.crc32(str(tmp_bytes))#python2.6 crc32() argument 1 must be string or read-only buffe
        self._file.write(bytearray(struct.pack('!i', crc)))

        # Write the end chunk. Start with the size.
        self._file.write(bytearray(struct.pack('!i', 0)))
        # Then the chunk type.
        self._file.write(bytearray(self._END_CHUNK_TYPE))

        tmp_bytes = bytearray()
        tmp_bytes.extend(bytearray(self._END_CHUNK_TYPE))
        #tmp_bytes.extend(self._bytes_to_hide)
        crc = binascii.crc32(str(tmp_bytes))
        self._file.write(bytearray(struct.pack('!i', crc)))

        #print 'Punk chunk injected'

    def _read_next_chunk(self):

        chunk_size = self._read_bytes_as_int(4)
        if self._mode == 'read':
            print 'Chunk size:', chunk_size

        chunk_type = self._read_bytes_as_ascii(4)
        if self._mode == 'read':
            print 'Chunk type:', chunk_type

        if self._mode == 'encode' and chunk_type == self._PUNK_CHUNK_TYPE:
            print 'already has something, use \'-m c\' first'
            return
        if self._mode == 'encode' and chunk_type == self._END_CHUNK_TYPE:

            self._inject_punk_chunk()
            self._file.close()

            return
        if self._mode == 'read' and chunk_type == self._END_CHUNK_TYPE:

            crc = self._read_bytes_as_hex(4)
            print 'CRC:', crc

            return

        if self._mode == 'del' and chunk_type == self._END_CHUNK_TYPE:

            self._TEMP_BYTES.extend(bytearray(struct.pack('!i', 0)))
            self._TEMP_BYTES.extend(bytearray(self._END_CHUNK_TYPE))
            self._TEMP_BYTES.extend(bytearray(binascii.a2b_hex('AE426082')))

            return
        if chunk_type == self._END_CHUNK_TYPE:
            return

        content = self._read_bytes(chunk_size)

        crc = self._read_bytes_as_hex(4)
        if self._mode == 'read':
            print 'CRC:', crc
            print ''

        if self._mode == 'del' and chunk_type != self._TEMP_CHUNK:
            self._TEMP_BYTES.extend(bytearray(struct.pack('!i', chunk_size)))
            self._TEMP_BYTES.extend(bytearray(chunk_type))
            self._TEMP_BYTES.extend(bytearray(content))
            self._TEMP_BYTES.extend(bytearray(binascii.a2b_hex(crc)))

        if self._mode == 'decode' and chunk_type == self._PUNK_CHUNK_TYPE:

            self._output.write(bytearray(content))
            self._output.close()
            self._file.close()

            return
        if self._mode == 'get' and chunk_type == self._PUNK_CHUNK_TYPE:

            self._file.close()
            self._TEMP_BYTES.extend(bytearray(content))

            return

        self._read_next_chunk()
        self._read_next_chunk()


    def read_png(self, png_filename):

        self._mode = 'read'
        self._file = open(png_filename, 'rb+')

        #the very first 8 bytes
        self._read_bytes(8)
        self._read_next_chunk()
    def get_chunk(self, png_filename):

        self._mode = 'get'
        self._file = open(png_filename, 'rb+')

        #the very first 8 bytes
        self._read_bytes(8)
        self._read_next_chunk()

        return self._TEMP_BYTES

    def del_chunk(self, png_filename, chunk):

        self._mode = 'del'
        self._file = open(png_filename, 'rb+')
        self._TEMP_CHUNK = chunk
        self._TEMP_BYTES.extend(bytearray(binascii.a2b_hex('89504E470D0A1A0A')))

        #the very first 8 bytes
        self._read_bytes(8)
        self._read_next_chunk()

        outfile = open('C'+png_filename, 'wb+')
        print 'New png: '+'C'+png_filename
        outfile.write(bytearray(self._TEMP_BYTES))
        outfile.close()
        self._file.close()


def encrypt_pass(passs, salt):
    hashlib.sha1(salt+passs).hexdigest()
def compare_pass(o, i):
    return o == hashlib.sha1(i).hexdigest()

def get_pass(p):

    import hashlib
    return hashlib.sha256(p).digest()[:16]


def encrypt(data, p):

    x = xtea.new(get_pass(p+_DEFAULT_SALT), IV=_DEFAULT_IV)
    if len(data) % 16 != 0:
        data += ' ' * (16 - len(data) % 16)
    z = x.encrypt(data)
    return z
```

```
def decrypt(data, p):

    x = xtea.new(get_pass(p+_DEFAULT_SALT), IV=_DEFAULT_IV)
    #data = data.decode('hex')
    return x.decrypt(data)



_DEFAULT_PASS = '1231#3aQ'
_DEFAULT_SALT = '!@#A'
_DEFAULT_IV = 'ABCDEFGa'#must be 8 bytes

usage = "usage: %prog -m a/e/c -s test.png -d datafiletohide [-p password]"#选项 + 参数 参数为没有类似-d的值，

parser = OptionParser(usage, version="%prog 1.0")

parser.add_option("-m", "--mode", dest="mode",
                  help="add/extract/clean data to/from png")
parser.add_option("-s", "--png",type="string",dest="png",
                  metavar="FILE", help="png filepath to hide data in")
parser.add_option("-d", "--data",type="string",dest="data",
                  metavar="FILE", help="data filepath")
parser.add_option("-p", "--password",type="string",dest="psw",
                  help="password to encrypt hideData")
parser.add_option("-r", "--read", action="store_true",dest="read",
                 default=False,help="show png structure")

try:
    import xtea
except Exception,e:
    print e
    import sys
    sys.exit()
#解析参数
(options, args) = parser.parse_args()#默认使用sys.argv[:1]当参数

if options.read:
    if options.png:
        punk = Punk()
        punk.read_png(options.png)
        import sys
        sys.exit()

if not (options.mode and options.png):
    import sys
    parser.print_help()
    sys.exit()

punk = Punk()
#must set
if options.mode == 'a':
    print '...'
    if options.data:
        if options.psw:
            dataencrypt = encrypt(open(options.data,'rb').read(), options.psw)
        else:
            dataencrypt = encrypt(open(options.data,'rb').read(), _DEFAULT_PASS)
        punk.encode(options.png, dataencrypt)
        print 'Done.'
    else:
```

```
    else:
        parser.error("use -d datafilepath to add to png")

elif options.mode == 'e':
    print '...'
    if options.psw:
        dataplain = decrypt(punk.get_chunk(options.png), options.psw).rstrip()
    else:
        dataplain = decrypt(punk.get_chunk(options.png), _DEFAULT_PASS).rstrip()
    if options.data:
        output = open(options.data, 'wb+')
    else:
        output = open('output', 'wb+')
    output.write(dataplain)
    output.close()
    print 'Done.'

elif options.mode == 'c':
    print '...'
    punk.del_chunk(options.png, punk._PUNK_CHUNK_TYPE)
    print 'Done.'

else:
    parser.print_help()
    sys.exit()

# aa = encrypt('abc', '123')

# print aa

# print len(decrypt(aa, '123').rstrip())

# punk = Punk()
# #punk.encode('1.png', open('BIW-IPAD-DEV.ipa', 'rb').read())
# #punk.decode('1.png', '3.png')
# #punk.read_png('C1.png')
# punk.del_chunk('C1.png', 'puNk')
```

参考

http://blog.brian.jp/python/png/2016/07/07/file-fun-with-pyhon.html