png隐写

原创

CodeStarr 已于 2022-03-01 18:49:09 修改 121 收藏

分类专栏： bin 开发 文章标签： go

于 2022-03-01 17:54:03 首次发布

文章目录

隐写：将信息植入其它数据，并可以提取出来。

仅关键逻辑提供源码。

1. PNG格式

PNG官网介绍了png的文件格式，可供参考。

也可以用010editor的png模板。

简述一下，首先是8字节magic:

```
89 50 4E 47 0D 0A 1A 0A
```

直接用uint64表示即可：

```
type Header struct {
 magic uint64 //  0:8
}
```

然后是块（chunk）数组，以type==“IHDR”开头，type=="IEND"结尾，中间是N个type=="IDAT"的块。

chunk结构（010editor模板）：

```
local uint32 CHUNK_CNT = 0;

// Generic Chunks
typedef struct {
    uint32  length;                     // Number of data bytes (not including length,type, or crc)
    local int64 pos_start = FTell();
    CTYPE   type <fgcolor=cDkBlue>;        // Type of chunk
    if (type.cname == "IHDR")
        PNG_CHUNK_IHDR    ihdr;
    else if (type.cname == "tEXt")
        PNG_CHUNK_TEXT    text;
    else if (type.cname == "PLTE")
        PNG_CHUNK_PLTE    plte(length);
    else if (type.cname == "cHRM")
        PNG_CHUNK_CHRM    chrm;
    else if (type.cname == "sRGB")
        PNG_CHUNK_SRGB    srgb;
    else if (type.cname == "iEXt")
        PNG_CHUNK_IEXT    iext(length);
    else if (type.cname == "zEXt")
        PNG_CHUNK_ZEXT    zext(length);
    else if (type.cname == "tIME")
        PNG_CHUNK_TIME    time;
    else if (type.cname == "pHYs")
        PNG_CHUNK_PHYS    phys;
    else if (type.cname == "bKGD")
        PNG_CHUNK_BKGD    bkgd(chunk[0].ihdr.color_type);
    else if (type.cname == "sBIT")
        PNG_CHUNK_SBIT    sbit(chunk[0].ihdr.color_type);
    else if (type.cname == "sPLT")
        PNG_CHUNK_SPLT    splt(length);
    else if (type.cname == "acTL")
        PNG_CHUNK_ACTL    actl;
    else if (type.cname == "fcTL")
        PNG_CHUNK_FCTL    fctl;
    else if (type.cname == "fdAT")
        PNG_CHUNK_FDAT    fdat;
    else if( length > 0 )
        ubyte   data[length];       // Data (or not present)
    local int64 data_size = FTell() - pos_start;
    uint32  crc <format=hex, fgcolor=cDkPurple>;  // CRC (not including length or crc)
    local uint32 crc_calc = Checksum(CHECKSUM_CRC32, pos_start, data_size);
    if (crc != crc_calc) {
        local string msg;
        SPrintf(msg, "*ERROR: CRC Mismatch @ chunk[%d]; in data: %08x; expected: %08x", CHUNK_CNT, crc, crc_calc
);
        error_message( msg );
    }
    CHUNK_CNT++;
} PNG_CHUNK <read=readCHUNK>;
```

struct解释：

```
type Chunk struct {
 Size uint32
 Type uint32
 Data []byte
    CRC  uint32 // crc32.ChecksumIEEE(type+data)
}
```

2. 功能设计

1. 最基本的，输入（-i）和输出（-o）；
2. 浏览各个chunk信息（–meta），因为每个块的data比较大，可以忽略（–suppress）；
3. 指定要添加的payload（–payload）,以及偏移（–offset）, 并指定启用的参数开关（–inject）；
4. 可以把payload注入到指定类型的块（–type）;
5. 加解密（–encode, --decode）以及密钥（–key）

当然payload是放在文件的最后（offset == file size）

```go
// main.go
// main.exe -i in.png -o out.png --inject --offset 0x85258 --payload 1234
// Encode: main.exe -i in.png -o encode.png --inject --offset 0x85258 --payload 1234 --encode --key secret
// Decode: main.exe -i encode.png -o decode.png --offset 0x85258 --decode --key secret

func init() {
 flags.StringVarP(&opts.Input, "input", "i", "", "Path to the original image file")
 flags.StringVarP(&opts.Output, "output", "o", "", "Path to output the new image file")
 flags.BoolVarP(&opts.Meta, "meta", "m", false, "Display the actual image meta details")

 flags.BoolVarP(&opts.Suppress, "suppress", "s", false, "Suppress the chunk hex data (can be large)")
 flags.StringVar(&opts.Offset, "offset", "", "The offset location to initiate data injection")
 flags.BoolVar(&opts.Inject, "inject", false, "Enable this to inject data at the offset location specified")
 flags.StringVar(&opts.Payload, "payload", "", "Payload is data that will be read as a byte stream")
 flags.StringVar(&opts.Type, "type", "rNDm", "Type is the name of the Chunk header to inject")
 flags.StringVar(&opts.Key, "key", "", "The enryption key for payload")
 flags.BoolVar(&opts.Encode, "encode", false, "XOR encode the payload")
 flags.BoolVar(&opts.Decode, "decode", false, "XOR decode the payload")
 flags.Lookup("type").NoOptDefVal = "rNDm"
 flags.Usage = usage
 flags.Parse(os.Args[1:])

 if flags.NFlag() == 0 {
  flags.PrintDefaults()
  os.Exit(1)
 }
 if opts.Input == "" {
  log.Fatal("Fatal: The --input flag is required")
 }
 if opts.Offset != "" {
  byteOffset, _ := strconv.ParseInt(opts.Offset, 0, 64)
  opts.Offset = strconv.FormatInt(byteOffset, 10)
 }
 if opts.Suppress && (opts.Meta == false) {
  log.Fatal("Fatal: The --meta flag is required when using --suppress")
 }
 if opts.Meta && (opts.Offset != "") {
  log.Fatal("Fatal: The --meta flag is mutually exclusive with --offset")
 }
 if opts.Inject && (opts.Offset == "") {
  log.Fatal("Fatal: The --offset flag is required when using --inject")
 }
 if opts.Inject && (opts.Payload == "") {
  log.Fatal("Fatal: The --payload flag is required when using --inject")
 }
 if opts.Inject && opts.Key == "" {
  fmt.Println("Warning: No key provided. Payload will not be encrypted")
 }
 if opts.Encode && opts.Key == "" {
  log.Fatal("Fatal: The --encode flag requires a --key value")
 }
}
```

3. 读取并解析png

图片应该不算是小文件了。bufio库可以通过缓冲区，降低访问本地磁盘的次数，进而提高文件读写的效率。

处理流程：os.File–>bufio.Reader–>bytes.Reader。如果省略bufio，理论上应该会慢一些。

```
func PreProcessImage(dat *os.File) (*bytes.Reader, error) {
 stats, err := dat.Stat()
 if err != nil {
  log.Fatal(err)
 }

 var size = stats.Size()
 b := make([]byte, size)

 bufR := bufio.NewReader(dat)
 if _, err := bufR.Read(b); err != nil {
  log.Fatal(err)
 }

 bReader := bytes.NewReader(b)

 return bReader, err
}
```

chunk结构上面已经定义了，但为了读写的时候确定偏移，再定义一个子类：

```
//MetaChunk inherits a Chunk struct
type MetaChunk struct {
 Chk    Chunk
 Offset int64
}
```

验证magic

最先读取的当然是magic头部。需要注意的是字节序。我们希望uint64的第一个字节（地址最高的字节）是文件的第一个字节，所以应该是大端。

后续的读写操作都是大端。

```
const (
 uint64Magic  = 0x89504e470d0a1a0a
)

func (mc *MetaChunk) validate(b *bytes.Reader) {
 var header Header

 if err := binary.Read(b, binary.BigEndian, &header.magic); err != nil {
  log.Fatal(err)
 }

 if header.magic == uint64Magic {
  fmt.Println("Valid PNG so let us continue!")
 }
}
```

解析

传参时注意要用*bytes.Reader指针类型，学过c指针的都能理解原因。

```
func (metaChunk *MetaChunk) ProcessImage(pBytesReader *bytes.Reader, cmdlineOpts *models.CmdLineOpts) {

    metaChunk.validate(pBytesReader)

    if cmdlineOpts.Meta {
  count := 1 //Start at 1 because 0 is reserved for magic byte
  var chunkType string
  for chunkType != endChunkType {
   metaChunk.getOffset(pBytesReader) // 获取当前文件指针偏移再读
   metaChunk.readChunk(pBytesReader)
   fmt.Println("---- Chunk # " + strconv.Itoa(count) + " ----")
   fmt.Printf("Chunk Offset: %#02x\n", metaChunk.Offset)
   fmt.Printf("Chunk Length: %s bytes\n", strconv.Itoa(int(metaChunk.Chk.Size)))
   fmt.Printf("Chunk Type: %s\n", metaChunk.chunkTypeToString())
   fmt.Printf("Chunk Importance: %s\n", metaChunk.checkCritType())

   if cmdlineOpts.Suppress == false {
    fmt.Printf("Chunk Data: %#x\n", metaChunk.Chk.Data)
   } else if cmdlineOpts.Suppress {
    fmt.Printf("Chunk Data: %s\n", "Suppressed")
   }
   fmt.Printf("Chunk CRC: %x\n", metaChunk.Chk.CRC)
   chunkType = metaChunk.chunkTypeToString()
   count++
  }
 }

}
```

main

```
func main() {
 dat, err := os.Open(opts.Input)
 defer dat.Close()
 bReader, err := utils.PreProcessImage(dat)
 if err != nil {
  log.Fatal(err)
 }
 png.ProcessImage(bReader, &opts)
}
```

4. 编码payload

注入文件则最终需要写入新文件。需要注意的是，chunk结构体应按照大端序写进一块buf，再写入文件：

```
func (mc *MetaChunk) marshalData() *bytes.Buffer {
 bytesMSB := new(bytes.Buffer)
 if err := binary.Write(bytesMSB, binary.BigEndian, mc.Chk.Size); err != nil {
  log.Fatal(err)
 }
 if err := binary.Write(bytesMSB, binary.BigEndian, mc.Chk.Type); err != nil {
  log.Fatal(err)
 }
 if err := binary.Write(bytesMSB, binary.BigEndian, mc.Chk.Data); err != nil {
  log.Fatal(err)
 }
 if err := binary.Write(bytesMSB, binary.BigEndian, mc.Chk.CRC); err != nil {
  log.Fatal(err)
 }

 return bytesMSB
}
```

写文件：

```
// reader: original file
// cmdlineOpts: provide the original file size (-paylaod size)
// arrBytesData: payload
func WriteData(reader *bytes.Reader, cmdlineOpts *models.CmdLineOpts, arrBytesData []byte) {
 offset, err := strconv.ParseInt(cmdlineOpts.Offset, 10, 64)
 if err != nil {
  log.Fatal(err)
 }

 fileWriter, err := os.OpenFile(cmdlineOpts.Output, os.O_RDWR|os.O_CREATE, 0777)
 if err != nil {
  log.Fatal("Fatal: Problem writing to the output file!")
 }
 reader.Seek(0, 0)

 var buff = make([]byte, offset)
 reader.Read(buff)
 fileWriter.Write(buff)
 fileWriter.Write(arrBytesData)
 if cmdlineOpts.Decode {
  reader.Seek(int64(len(arrBytesData)), 1) // right bitshift to overwrite encoded chunk
 }
 _, err = io.Copy(fileWriter, reader)
 if err == nil {
  fmt.Printf("Success: %s created\n", cmdlineOpts.Output)
 }
}
```

完善解析：

```
func (metaChunk *MetaChunk) ProcessImage(pBytesReader *bytes.Reader, cmdlineOpts *models.CmdLineOpts) {

    metaChunk.validate(pBytesReader)

    if (cmdlineOpts.Offset != "") && cmdlineOpts.Encode {
  var newChunk MetaChunk
  newChunk.Chk.Data = utils.XorEncode([]byte(cmdlineOpts.Payload), cmdlineOpts.Key)
  newChunk.Chk.Type = newChunk.strToInt(cmdlineOpts.Type)
  newChunk.Chk.Size = newChunk.createChunkSize()
  newChunk.Chk.CRC = newChunk.createChunkCRC()
  buffer := newChunk.marshalData()
  arrBytes := buffer.Bytes()
  fmt.Printf("Payload Original: % X\n", []byte(cmdlineOpts.Payload))
  fmt.Printf("Payload Encode: % X\n", newChunk.Chk.Data)
  utils.WriteData(pBytesReader, cmdlineOpts, arrBytes)
 }
    // ...

}
```

解码部分略。