极验验证码破解

[国家企业信用信息公示系统为例]

[2017.08.01]

殷昊
[苏州大学]

目录

一、网站 http://www.gsxt.gov.cn 滑动验证码概述

（对滑动验证码了解的朋友请跳过本章节）

1. 浏览器打开目标站点，将会出现图示搜索框。

2. 在搜索框中输入待查询企业名（如：百度），点击查询，将会出现如图所示验证码。

上述滑动验证码即为 http://www.geetest.com/（极验验证）所提供的验证码服务。该滑动验证码号称利用机器学习和神经网络构建线上线下的多重静态、动态防御模型，具体描述可参见 http://www.geetest.com/feature.html。

3. 拖动验证码中的滑块，首先看失败时的效果。

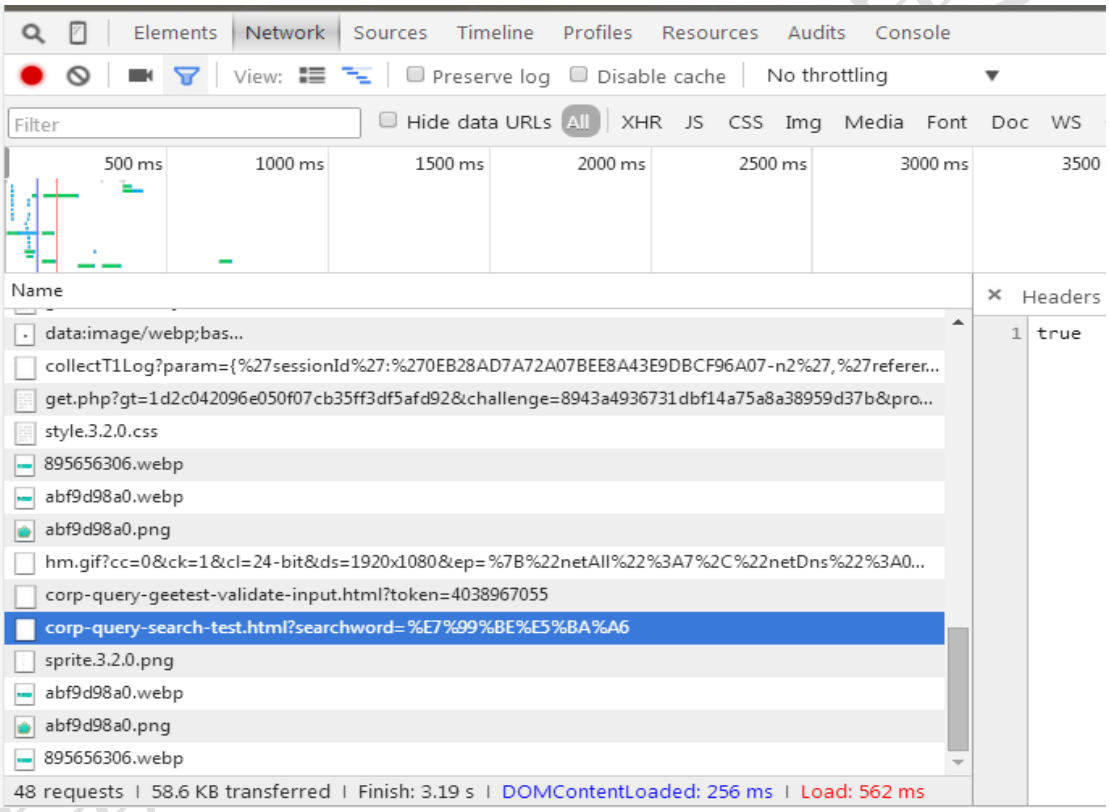
拖动滑块到缺口位置，验证成功效果如下：

4. 以上为对目标网站滑动验证码的一些基本描述，下面章节将会详细讲解如何使用计算机程序（爬虫）来对该验证码进行破解。

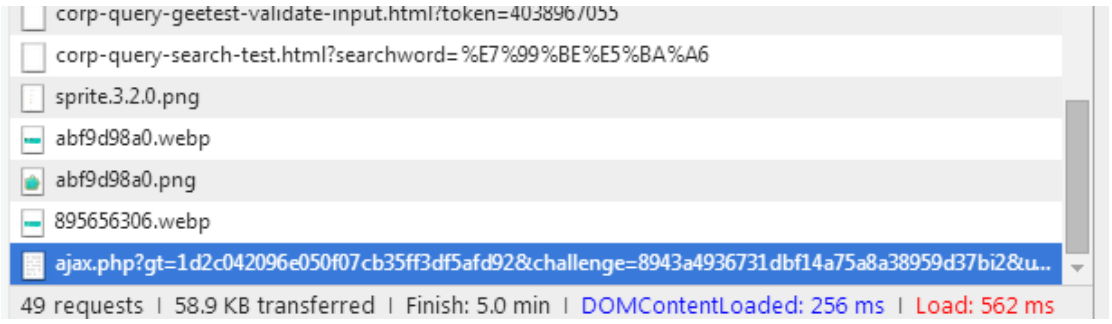
二、极验验证码破解-抓包分析

1. 为了分析服务器如何验证客户端是否成功拖动滑块至缺口（即完成验证），我们启动浏览器（我使用的是 360 安全浏览器）-工具-开发人员工具（F12），在输入框输入“百度”，点击查询。在开发者工具-Network 中，我们可以看到如下数据包。

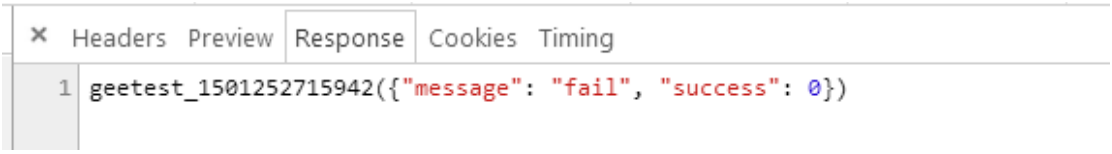
   （PS:若数据包的数量较少，可先启动开发者工具 F12，刷新页面）



2. 我们象征性的拖动滑块，可以看见 Network 选项下多出一个数据包 ajax.php?gt=…&challenge=…,如下图所示：

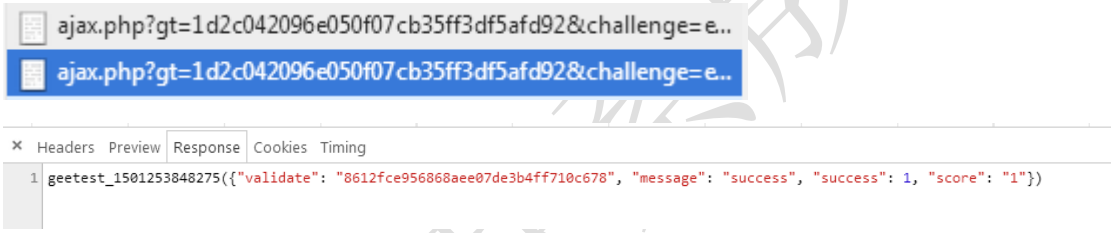


鼠标点击上述数据包，可在右侧查看该数据包的详细信息。点击 Response 可查看服务器响应信息，如下所示：



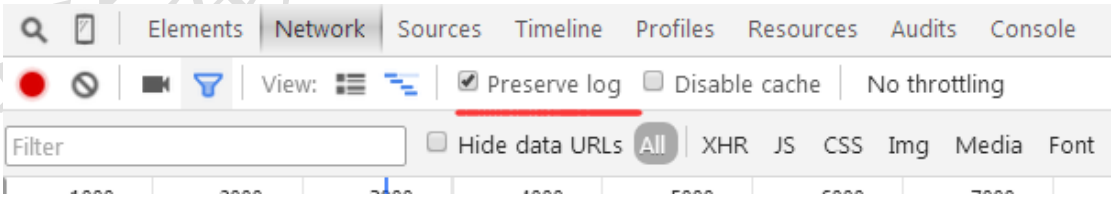
显然，服务器返回该验证码验证失败：fail。

3. 为了查看验证成功时的服务器响应信息，我们拖动滑块至缺口处，已同样方式查看最新的 ajax.php 页面，我们发现此时 ajax.php?数据包的 Response 变为如下信息：



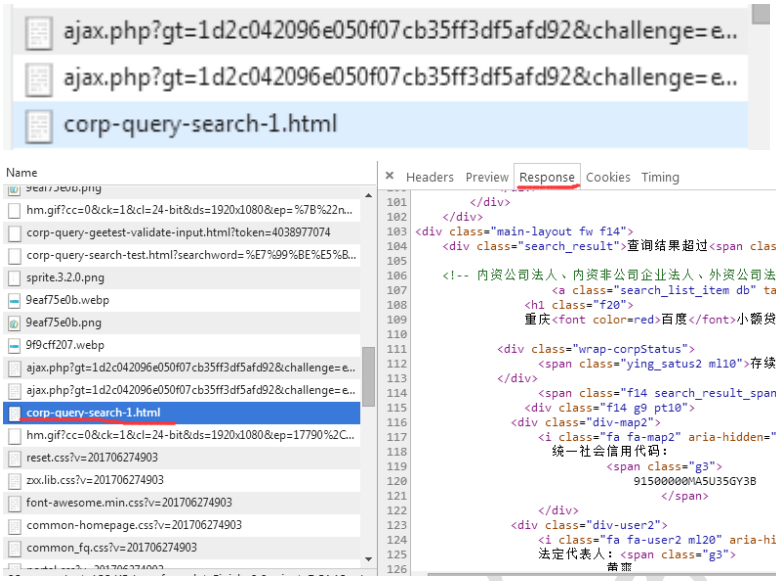
有上述可知，当服务器检测到客户端验证成功时，返回 validate(可看作是一串长字符串密码)和其他 success 信息。

（PS：为防止验证成功时页面快速切换至查询结果页面，可在 Network 选项卡下将 Preserve log 选项勾选，如下图所示：）



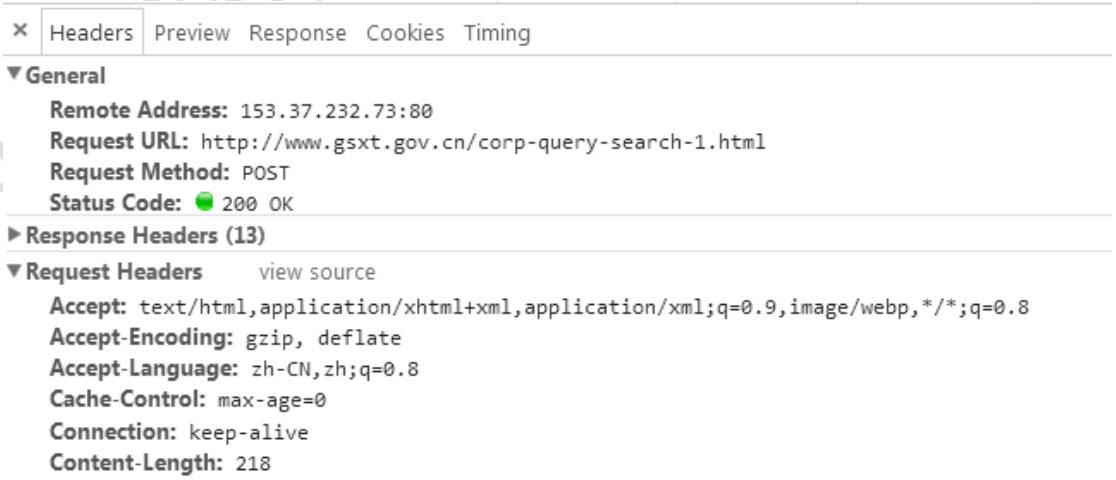
4. 当验证成功时，页面跳转至查询结果页面，如下所示：

通过分析 Network 下的数据包可以发现，在最新的验证成功的 ajax.php? 数据包下面，有一个 crop-query-search-1.html 数据包，如下所示：



通过查看该项的 Response，我们可以发现，该 Response 的内容即为浏览器页面上展示的内容，亦即我们所需爬取的内容。

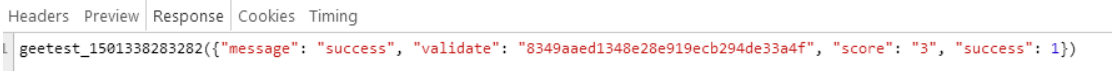
为了获取该 Response 内容，我们将模拟请求该 url，我们查看该请求的头部 headers 信息，如下所示：



从 Headers 中我们发现，该请求为 POST 请求，同时发送 Form Data 给服务器。接下来我们将分析该 Form Data 是如何得到的，其中 geetest_challenge 和 geetest_validate 是关键信息，geetest_seccode 是由 geetest_validate 后加上“|jordan”。Form Data 相当于密码，浏览器通过向服务器请求页面，并发送 Form Data，服务器端验证该密码正确，将返回页面信息。

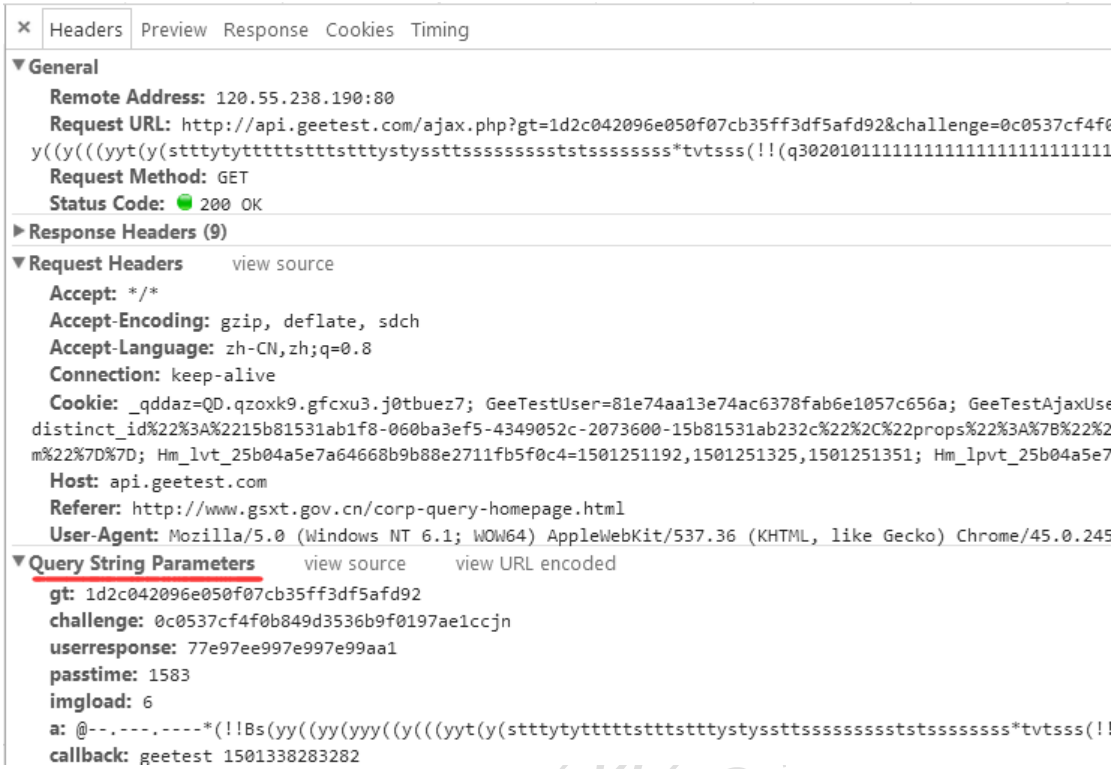
接下来我们将逆向探索如何得到 **geetest_challenge** 和 **geetest_validate**。（实验证明：tab 可为空或’ent_tab’，不影响结果，token 可设一相同长度的随机数，不影响结果，searchword 为查询关键词）

5. 回忆上述 3 中验证成功时，ajax.php?的 Response 信息可知，validate 的值是由该请求获得。



我们将模拟该请求，以获得 validate 信息，构造 geetest_validate 和 geetest_seccode。

6. 为了模拟请求 ajax.php?，我们查看该数据包的 Headers 信息：

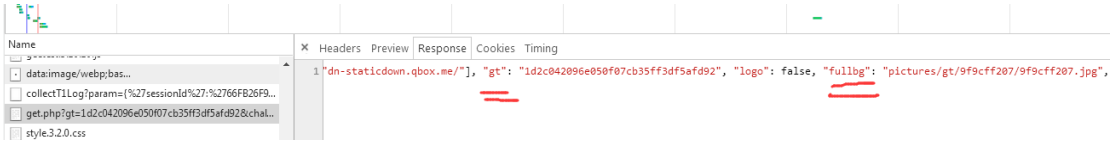


从上图可以看出，此请求附带 7 个参数。首先，我们可以大胆猜测 passtime 应该是拖动验证码的时间，imgload 应该是验证码图片载入时间，callback 应该是“geetest_”+当前时间的时间戳，如下：

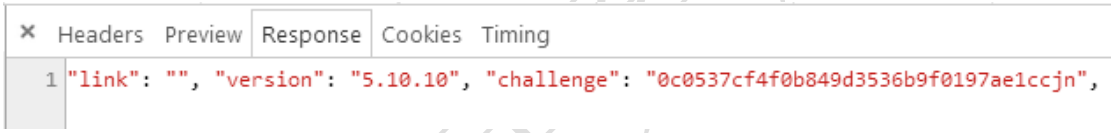
关于 passtime，我们知道验证码图片在不同的机器上受网速影响，加载时间是不同的，因此该项可暂考虑设为一个 0-1000ms 的随机数或固定值。

7. 我们知道极验验证是根据用户拖动鼠标的轨迹来判断是人为操作还是机器的。Userresponse 和 imgload 从参数名上可猜测是跟拖动滑块的轨迹（用户行为）来生成的。参数 a 最为复杂，我们先看 gt 和 challenge 是如何生成的。

8. 分析 Network 下的数据包（ajax.php？之前的包），我们发现除去一些 css、png 等无关紧要的东西外，有个 get.php?gt=…还是挺显眼的，大胆点进去，如下：

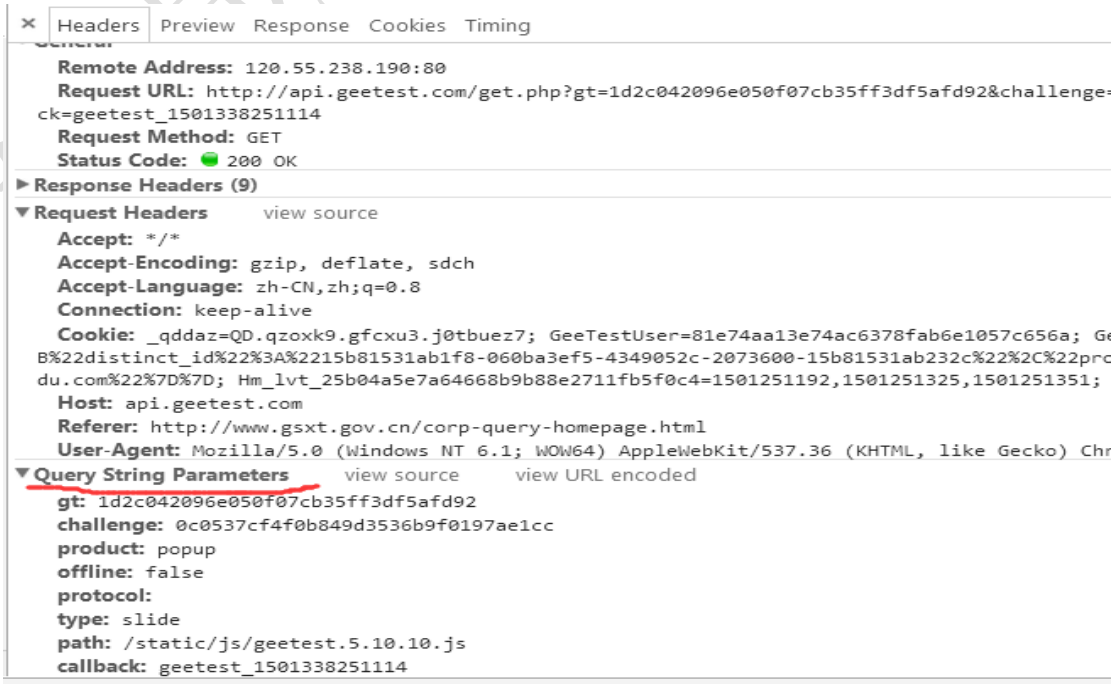


从该数据包的 Response 中可以发现，"gt": …，对比发现，这就是我们要的 gt 啊啊啊！“fullbg”也暗示着我们，它表示验证码的背景图片。此时，可以感觉到该 Response 的信息量之大，继续往后查看，可以发现：



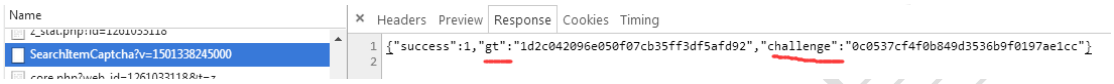
“challenge”出现了，对比发现，这也是我们要的 challenge!!!

9. 为了得到上述大量信息，我们模拟请求 get.php?，查看该包的 Headers 如下：



我们发现，该请求的参数还有一个 gt 和 challenge(真实一个接着一个…)。除了 gt 和 challenge,其它 6 个参数看起来没什么价值，可直接保留，我们来分析 gt 和 challenge 的生成。

继续查看 Network 中的数据包（get.php?之前的），我们可以发现 SearchItemCaptcha?v=…这个包的 Response 如下：



惊呆.jpg！比较发现这里的 gt 和 challenge 就是我们需要的值。

双击 

可以打开页面，刷新发现每次页面展示的返回值中 challenge 都在变化。这是因为该 api 的参数 v 是根据请求时间动态变化的，因此我们也可以将 v 设为当前时间的时间戳。

10. OK！至此我们可以按照 SearchItemCaptcha?v=…➔ get.php? ➔ ajax.php？ ➔crop-query-search-1.html 的顺序请求数据。关键问题是如何构造 ajax.php？参数中的 userresponse、passtime 和 a。passtime 从名称可以看出是拖动滑块的时间，userresponse 和 a 应该是根据拖动滑块的轨迹进行加密的。接下来我们将在本地搭建一个简易的 web server，修改相关 js 源码，使得拖动滑块时输出鼠标轨迹信息 Track。并分析 js 源码，找出根据 Track 得到的 userresponse 和 a 的算法。

三、极验验证码破解-搭建本地验证码服务

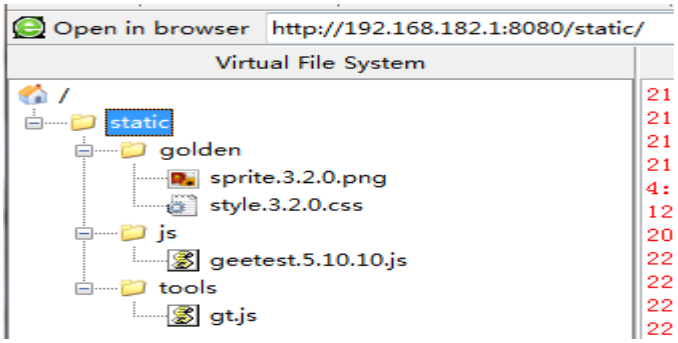
1. https://github.com/GeeTeam/gt-python-sdk 从上述链接下载 SDK 并按教程安装。

2. 打开 static/index.html。

```
63    <!-- 为使用方便，直接使用jquery.js库 -->
64    <script src="http://libs.baidu.com/jquery/1.9.0/jquery.js"></script>
65    <!-- 引入封装了failback的接口--initGeetest -->
66    <script src="http://static.geetest.com/static/tools/gt.js"></script>
```

可以发现，该页面请求的 gt.js 是来自 geetest 远程服务器的。我们要想修改 js 源码，打印相关信息，必须使得请求的 js 来自本地，因此我们搭建简单的文件服务器。

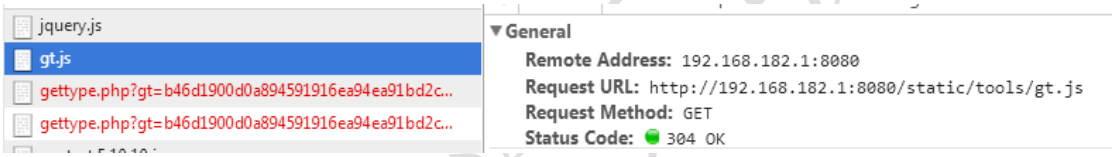
3. https://pan.baidu.com/s/1c26btBE 从上述链接下载 HFS 软件。运行软件，按图示搭建文件夹（所需文件从 Network 下载）：



修改 index.html 中 gt.js 的来源，如下所示(地址为 hfs 的地址)：

```
<!-- 为使用方便， 直接使用jquery.js库 -->
<script src="http://libs.baidu.com/jquery/1.9.0/jquery.js"></script>
<!-- 引入封装了failback的接口--initGeetest -->
<script src="http://192.168.182.1:8080/static/tools/gt.js"></script>
```

从 http://localhost:8000/ 下抓包发现，gt.js 的确来自本地 hfs：



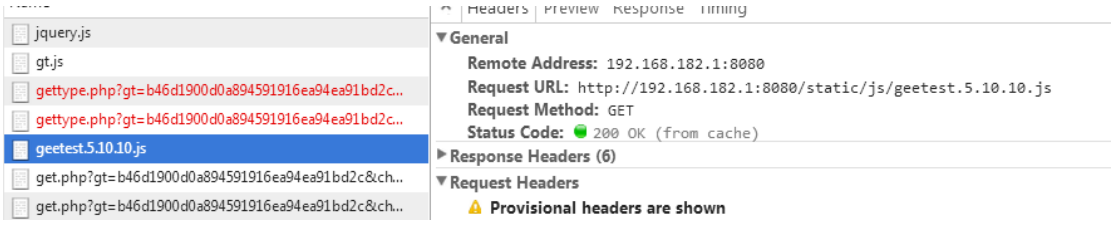
4. 打开 gt.js。

```
    fallback_config: {
        slide: {
            static_servers: ["static.geetest.com", "dn-staticdown.qbox.me"],
            type: 'slide',
            slide: '/static/js/geetest.0.0.0.js'
        },
        fullpage: {
            static_servers: ["static.geetest.com", "dn-staticdown.qbox.me"],
            type: 'fullpage',
            fullpage: '/static/js/fullpage.0.0.0.js'
        }
    },
```

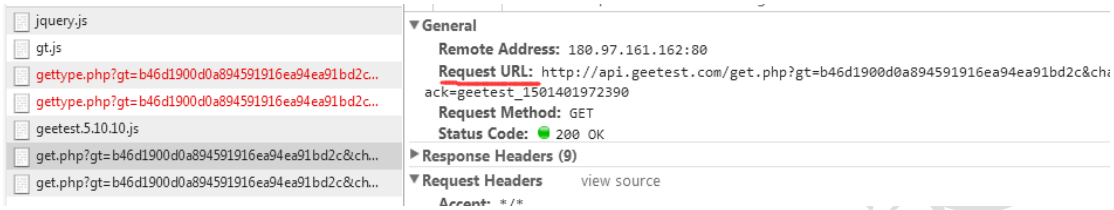
我们发现 geetest.js 是由 gt.js 调用，修改 static_server 为本地地址：

```
fallback_config: {
    slide: {
        static_servers: ["192.168.182.1:8080", "dn-staticdown.qbox.me"],
        type: 'slide',
        slide: '/static/js/geetest.5.10.10.js'
    },
    fullpage: {
        static_servers: ["192.168.182.1:8080", "dn-staticdown.qbox.me"],
        type: 'fullpage',
        fullpage: '/static/js/fullpage.0.0.0.js'
    }
```

抓包发现，geetest.js 的确来自本地 hfs：



（ps. 图片等其他本地没有的资源将会继续请求源服务器）。



至此，我们已搭建好本地验证码服务，下面我们将分析 geetest. js，找出相关参数 userresponse 和 a 的加密过程。

四、极验验证码破解-分析 geetest.js，得到所需参数

1. 为了找出如何生成 userresponse 和 a，我们在 geetest. js 中搜索 userrespnse，发现只有一处提到 userresponse，如下所示：

```
var p = {
    gt: a.config.gt,
    challenge: a.config.challenge,
    userresponse: ca.ra(l, a.config.challenge),
    passtime: Q.t("endTime", a.id).getTime() - Q.t("startTime", a.id),
    imgload: Q.t("imgload", a.id),
    a: encodeURIComponent(n)
};
```

可以看出 userresponse 和 a 都在上述代码中赋值。

首先，我们追踪 ca. ra，得到下面代码：

```
ca.ra = function(a, b) {
    console.log("b:"+b);
    for (var c = b.slice(32), d = [], e = 0; e < c.length; e++) {
        var f = c.charCodeAt(e);
        d[e] = f > 57 ? f - 87 : f - 48
    }
    c = 36 * d[0] + d[1];
    var g = Math.round(a) + c;
    b = b.slice(0, 32);
    var h, i = [[], [], [], [], []],
```

计算 userresponse 需要参数 l 和 challenge，challenge 我们已知，

下面分析 l（先猜测为滑块移动的距离）。

由 `a: encodeURIComponent(n)` 可得到 a 由 n 生成，

```
n = oa.qa(a.id); oa = function() {
    return {
        qa: f,
        na: b,
        c: a
    }
```

由上述可知 oa.qa 的返回值是 f，下面继续追踪 f：

```
    },
    f = function(a) {
        for (var b, f = c(Q.t("arr", a)), g =
        j++) b = e(f[j]),
        b ? h.push(b) : (g.push(d(f[j][0])), h
        i.push(d(f[j][2]));
        console.log("lala:"+c(Q.t("arr", a)).j
        console.log("a:"+g.join("") + "!!" + h
        return g.join("") + "!!" + h.join("")
    };
    return {
        qa: f,
        na: b,
        c: a
    }
```

由上述可知（console 为我添加，目的是打印信息），f 的返回值即为所求 a，观察其包含!! 信息，比较所需 a 值：

```
a: 6-.-/1313212200.-(!!4w((!)(!)!)((((((((((tssss(!!($)/11111111111111202028E$/n
```

可以发现，目前推测无误。

2. 我们发现 f 依赖于 oa 下的 c 函数，追踪 c 函数：

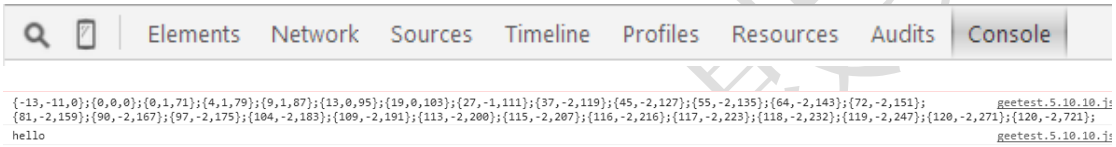
```
c = function(a) {
for (var b, c, d, e = [], f = 0, g = [], h
.round(a[h + 1][0] - a[h][0]),
c = Math.round(a[h + 1][1] - a[h][1]),
d = Math.round(a[h + 1][2] - a[h][2]),
```

观察 c 函数的参数 a，由 a[h+1][1]可知 a 应该是一个二维数组。

添加如下信息，输出 a 的内容：

```
c = function(a) {
    s=""
    for(i=0;i<a.length;i++)
    {
        s=s+"{"+a[i][0]+","+a[i][1]+","+a[i][2]+"};"
    }
    console.log(s);
    console.log("hello");
```

刷新 http://localhost:8000/, 查看 console 控件下的信息：



可以发现，这很大可能就是我们所要找的滑动轨迹 Track。

回到之前 userresponse 中的 l 参数，我们在相应位置输出：

```
var l = parseInt(h),
n = oa.qa(a.id);
console.log("a.id2:"+a.id+" l:"+l);
```

查看 Console 发现：

```
a.id2:1501424908989 l:120
```

之前我们猜测 l 为滑动距离，比较 l 与 Track 最后一个坐标，可以发现 l 与 Track[-1][0]相等，因此验证我们猜想—l 是滑动距离。

3. 要想得到 userresponse，需要得到 l，需要得到 Track, 由于 Track 是随机采样生成的离散轨迹坐标序列，因此我们需要根据缺口的位置来模拟 Track。由于 Track 较难模拟，我们先假设已根据缺口距离生成了 Track。

则可以写出生成 userresponse 的函数：

```
def cal_userresponse(a,b):
    d=[]
    c=b[32:]
    for e in range(len(c)):
        f=ord(str(c[e]))
        tmp=f-87 if f>57 else f-48
        d.append(tmp)

    c=36*d[0]+d[1]
    g=int(round(a))+c
    b=b[:32]

    i=[[],[],[],[],[]]
    j={}
    k=0
    e=0
    for e in range(len(b)):
        h=b[e]
        if h in j:
            pass
        else:
            j[h]=1
            i[k].append(h)
            k+=1
            k=0 if (k==5) else k
```

```
n=g
o=4
p=""
q=[1,2,5,10,50]
while n>0:
    if n-q[o]>=0:
        m=int(random.random()*len(i[o]))
        p+=str(i[o][m])
        n-=q[o]
    else:
        del(i[o])
        del(q[o])
        o-=1
return p
```

参数 a 和 b 分别是 l 和 challenge。上述函数是我根据

```
ca.ra = function(a, b) {
```

改写成 Python 形式

至此，userresponse 参数破解成功，接下来看 a 的生成。

4. 之前分析 a 是由 oa 下的 f 函数生成：

```
f = function(a) {
    for (var b, f = c(Q.t("arr", a)), g = [], h = [], i = [],
    j++) b = e(f[j]),
    b ? h.push(b) : (g.push(d(f[j][0])), h.push(d(f[j][1]))),
    i.push(d(f[j][2]));
    console.log("lala:"+c(Q.t("arr", a)).join(" "));
    console.log("a:"+g.join("") + "!!" + h.join("") + "!!" + i
    return g.join("") + "!!" + h.join("") + "!!" + i.join("")
};
return {
    qa: f,
    na: b,
    c: a
```

其中参数 a 是 Track。其中 f 函数用到了 c 函数、e 函数和 d 函数。

```
c = function(a) {

d = function(a) {

e = function(a) {
```

根据 js 函数源码，我将其改写成 python 函数。

```
# 计算每次间隔    相当于c函数
def fun_c(a):
    g=[]
    e=[]
    f=0
    for h in range(len(a)-1):
        b=int(round(a[h+1][0]-a[h][0]))
        c=int(round(a[h+1][1]-a[h][1]))
        d=int(round(a[h+1][2]-a[h][2]))
        g.append([b,c,d])
```

```
def fun_e(item):   # 相当于e函数
    b=[[1, 0], [2, 0], [1, -1], [1, 1], [0, 1], [0, -1], [3, 0], [2, -1], [2, 1]]
    c='stuvwxyz~'
    for i,t in enumerate(b):
        if t==item[:2]:
            return c[i]
    return 0
```

```
def fun_d(a):
    b='()*,-./0123456789:?@ABCDEFGHIJKLMNOPQRSTUVWXYZ_abcdefghijklmnopqr'
    c=len(b)
    d=''
    e=abs(a)
```

代码只截取部分，具体细节后面会开源。

```
def fun_f(track_list):
    skip_list=fun_c(track_list)
    g,h,i=[],[],[]
    for j in range(len(skip_list)):
        b=fun_e(skip_list[j])
        if b:
            h.append(b)
        else:
            g.append(fun_d(skip_list[j][0]))
            h.append(fun_d(skip_list[j][1]))
        i.append(fun_d(skip_list[j][2]))
    return ''.join(g)+'!!'+''.join(h)+'!!'+''.join(i)
```

完整的 f 函数如上所示，参数为 Track，返回值即为 f（也就是所需的 a）。

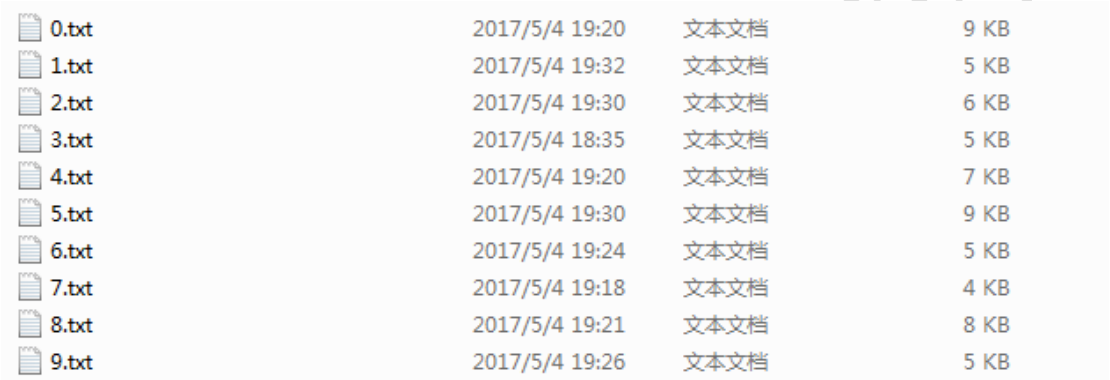
至此，参数 a 也破解完成!!! 只差最会一步，Track 如何得到。

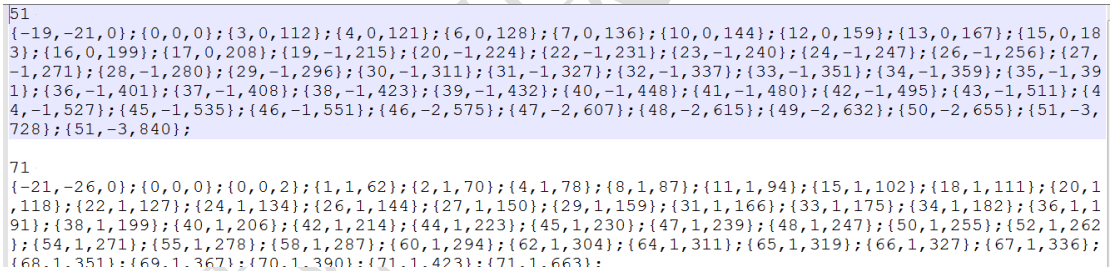
五、极验验证码破解-Track 的获取

1. Track 的生成可以根据图片缺口的距离 d，使用随机函数随机采样生成。比如 d=120,则我们控制总拖动时间为 t（t 一般小于 3s），则可以每很小的时间内（每次随机时间，如 10ms）移动一小段随机距离，最后在时刻 t 时正好移动到 d。但考虑人拖动会有先加速在减速的特点（或许还有其他特点），geetest 的服务器可能会识别到时机器所为，我们很难找到人行为的轨迹特点并且难以模拟，因此生成轨迹不太可行（也许可行，但代价较大）。
2. 我们提出另一种方法替代 Track 的生成，即手工事先存储备用 Track。验证码图片总长大概 250 左右，由于我们已经实现了在 Console 中打印 Track 的 js，因此我们可以多次刷新 http://localhost:8000/页面，得到不同缺口位置的验证码，手

动拖动至缺口处，保存 Console 中的 Track。

3. 经过试验发现，缺口位置大多停留在中间位置，并且拖动误差在 3 以内都可以接受。因此我们可以用当前位置 d 的 Track 来代替 d-1 和 d+1 的 Track（如缺口位置 120，则 119 和 121 的 Track 可以不用测试，直接使用 120 的 Track）。这样大大减少了刷新页面获取 Track 的次数，我收集的 Track 列表如下：





```
51
{-19,-21,0};{0,0,0};{3,0,112};{4,0,121};{6,0,128};{7,0,136};{10,0,144};{12,0,159};{13,0,167};{15,0,183};{16,0,199};{17,0,208};{19,-1,215};{20,-1,224};{22,-1,231};{23,-1,240};{24,-1,247};{26,-1,256};{27,-1,271};{28,-1,280};{29,-1,296};{30,-1,311};{31,-1,327};{32,-1,337};{33,-1,351};{34,-1,359};{35,-1,391};{36,-1,401};{37,-1,408};{38,-1,423};{39,-1,432};{40,-1,448};{41,-1,480};{42,-1,495};{43,-1,511};{44,-1,527};{45,-1,535};{46,-1,551};{46,-2,575};{47,-2,607};{48,-2,615};{49,-2,632};{50,-2,655};{51,-3,728};{51,-3,840};

71
{-21,-26,0};{0,0,0};{0,0,2};{1,1,62};{2,1,70};{4,1,78};{8,1,87};{11,1,94};{15,1,102};{18,1,111};{20,1,118};{22,1,127};{24,1,134};{26,1,144};{27,1,150};{29,1,159};{31,1,166};{33,1,175};{34,1,182};{36,1,191};{38,1,199};{40,1,206};{42,1,214};{44,1,223};{45,1,230};{47,1,239};{48,1,247};{50,1,255};{52,1,262};{54,1,271};{55,1,278};{58,1,287};{60,1,294};{62,1,304};{64,1,311};{65,1,319};{66,1,327};{67,1,336};{68,1,351};{69,1,367};{70,1,390};{71,1,423};{71,1,663};
```

为了方便统计，我按个位数将 Track 存放在 10 个文件中。

我们将所有的 Track 整合为 dict（t_dict.pkl），格式如下：

{k1:v1}，其中 k1 为缺口位置，v1 为 Track（字符串形式）。

```
cPickle.dump(t_dict,open('t_dict.pkl','w'))
```

至此我们得到 Track 备用列表，我们可以根据实际的缺口位置获得相应的 Track 值，下一节我们将会讲解如何得到缺口距离验证码左边的相对距离 d。

六、极验验证码破解-获取背景图片及缺口距离 d 的计算

1. 我们首先寻找图片的来源。回忆分析 get.php？的时候，看到过“fullbg”的出现，因此很大可能背景图片信息是通过 get.php？传来的。查看 Response 如下：

```
"bg": "pictures/gt/9f9cff207/bg/9f6cfe44a.jpg",
```

```
"fullbg": "pictures/gt/9f9cff207/9f9cff207.jpg",
```

根据图片的 url 打开图片：

可以发现图片已经乱码，这是因为返回的图片是局部重合产生的。查看验证码图片的审查元素：


```
<div class="gt_cut_fullbg gt_show">
  <div class="gt_cut_fullbg_slice" style="background-image: url(http://static.geetest.com/pictures/gt/26ebd36a0/26ebd36a0.webp); background-position: -157px -58px;"></div>
  <div class="gt_cut_fullbg_slice" style="background-image: url(http://static.geetest.com/pictures/gt/26ebd36a0/26ebd36a0.webp); background-position: -145px -58px;"></div>
  <div class="gt_cut_fullbg_slice" style="background-image: url(http://static.geetest.com/pictures/gt/26ebd36a0/26ebd36a0.webp); background-position: -265px -58px;"></div>
  <div class="gt_cut_fullbg_slice" style="background-image: url(http://static.geetest.com/pictures/gt/26ebd36a0/26ebd36a0.webp); background-position: -277px -58px;"></div>
  <div class="gt_cut_fullbg_slice" style="background-image: url(http://static.geetest.com/pictures/gt/26ebd36a0/26ebd36a0.webp); background-position: -181px -58px;"></div>
  <div class="gt_cut_fullbg_slice" style="background-image: url(http://static.geetest.com/pictures/gt/26ebd36a0/26ebd36a0.webp); background-position: -169px -58px;"></div>
  <div class="gt_cut_fullbg_slice" style="background-image: url(http://static.geetest.com/pictures/gt/26ebd36a0/26ebd36a0.webp); background-position: -241px -58px;"></div>
```

可以发现，展示的图片是从原始乱码图片中多次截取小段，合成而成的。具体的合成方式如 background-position 所示。

如 background-position：-157px,-58px。则该小段图片为源乱码图片的（157,58，157+10,58+58）。根据上述分析，我么可以还原 bg 和 fullbg 的非乱码图片（即所看见的背景图片）。

我们通过比较两张图片的像素值，即可得到缺口的位置，缺口左上角横坐标的值即为 d。我们封装了 get_dist 函数如下：

```
# 计算缺口距离
def get_dist(image1,image2):
    # 合并图片使用
    location_list=cPickle.load(open('location_list.pkl'))

    jpgfile1=cStringIO.StringIO(urllib2.urlopen(image1).read())
    new_image1=get_merge_image(jpgfile1,location_list)

    new_image1.save('image1.jpg')

    jpgfile2=cStringIO.StringIO(urllib2.urlopen(image2).read())
    new_image2=get_merge_image(jpgfile2,location_list)

    new_image2.save('image2.jpg')

    i=0
    for i in range(260):
        for j in range(116):
            if is_similar(new_image1,new_image2,i,j)==False:
                # 找到缺口    返回缺口的横坐标i
                # 因为图片是水平移动 所以不需要缺口的纵坐标
                return i
```

函数其它细节请参见项目源码。

七、极验验证码破解-总结

1. 至此，geetest 验证码的关键技术点已经讲解完，有没有感觉号称使用深度学习技术进行人机验证的滑动验证码也不过如此。最近我会在 Github 上开源所有代码，希望得到大家的指点。
2. 当时做这个项目大概断断续续做了两星期左右（2017.03），开始是参照网上教程使用 selenium 控制鼠标来实现，后来发现鼠标移动的速度太机械化，成功率太低（滑动到缺口处，但被识别为机器行为）。所以，我采取了退而求其次的方法，避免对轨迹路径的生成，直接使用已经成功验证的历史轨迹来作为当前轨迹。经过试验，这种方法成功率接近 100%，且复杂度不高，历史轨迹单独存在硬盘，可定期更新（以防止轨迹被封）。
3. 最近（2017.07）突然想写个文档教程分享技术，算是个学习笔记，也算是对自己曾经努力的记载。该文档断断续续写了 4 个晚上的时间，欢迎大家阅读并指正。
4. 最后附上 Github 地址，里面有一些小爬虫和 NLP 相关的项目，欢迎围观。https://github.com/FanhuaandLuomu

殷昊 苏州大学

Phone/WeChat:13771902647

Qq:1049755192

2017.08.01

参考文献

1. http://blog.csdn.net/paololiu/article/details/52514504

2. https://zhuanlan.zhihu.com/p/22866110?refer=windev

3. https://zhuanlan.zhihu.com/p/22404294