安卓脱壳&&协议分析&&burp辅助分析插件编写

前言

本文由 **本人** 首发于 先知安全技术社区： https://xianzhi.aliyun.com/forum/user/5274

前言

本文以一个 app 为例，演示对 app脱壳，然后分析其 协议加密和签名方法，然后编写 burp 脚本以方便后面的测试。

文中涉及的文件，脱壳后的 dex 都在：

链接: https://pan.baidu.com/s/1nvmUdq5 密码: isrr

对于 burp 扩展和后面加解密登录数据包工具的的源码，直接用 jd-gui 反编译 jar 包即可。

正文

首先下载目标 apk ，然后拖到 GDA 里面看看有没有壳。

发现是腾讯加固，可以通过修改 dex2oat 的源码进行脱壳。

```
//////////////////////////////分割线 以下为添加的代码//////////////////////////////////////////////////////////////////////////
      std::string dex_name = dex_file->GetLocation();
      LOG(INFO) << "haclh-->  dex2oat::dex_file name-->" << dex_name;

      LOG(INFO) <<"the dumpall_apk value:::"  << dumpall_apk;


      if (dumpall_apk == 1||
       dex_name.find("jiagu") != std::string::npos
        || dex_name.find("cache") != std::string::npos
        || dex_name.find("files") != std::string::npos
        || dex_name.find("tx_shell") != std::string::npos
        || dex_name.find("app_dex") != std::string::npos
        || dex_name.find("nagain") != std::string::npos) {

              int nDexLen = dex_file->Size();
              char pszDexFileName[260] = {0};
              sprintf(pszDexFileName, "%s_%d_dump_by_haclh", dex_name.c_str(), nDexLen);


              int fd = open(pszDexFileName, O_WRONLY | O_CREAT | O_TRUNC, S_IRWXU);
              if (fd > 0) {
                if (write(fd, (char*)dex_file->Begin(), nDexLen) <= 0) {
                  LOG(INFO) << "haclh--> dex2oat::write dex file failed-->" << pszDexFileName;
                } else {
                  LOG(INFO) << "haclh--> dex2oat::write dex file success-->" << pszDexFileName;
                }
                close(fd);
              } else {
                LOG(INFO) << "haclh--> dex2oat::open dex file failed-->" << pszDexFileName;
              }
      }
//////////////////////////////分割线 以上为添加的代码//////////////////////////////////////////////////////////////////////////
```

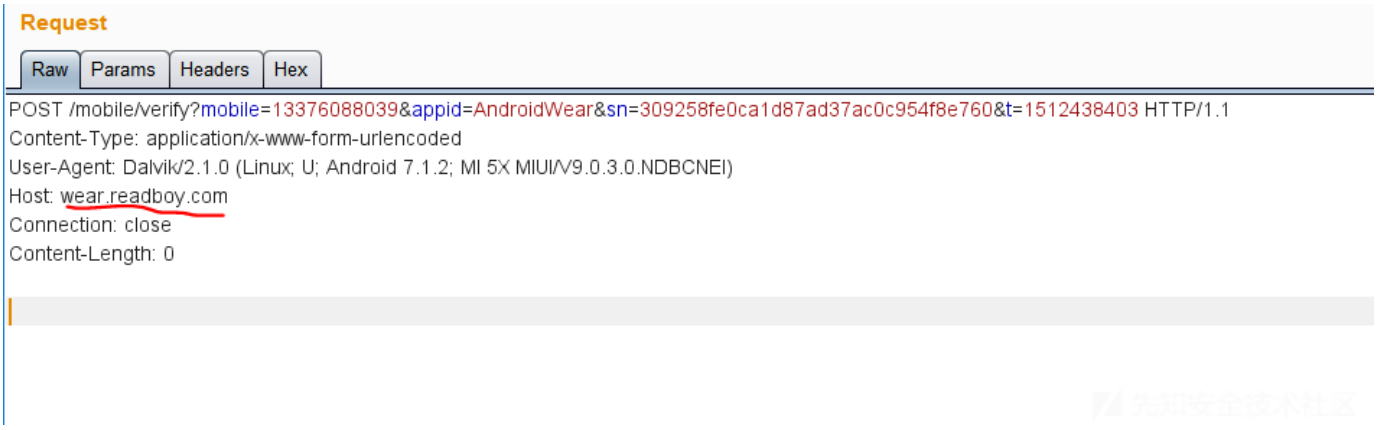
具体可以看: https://bbs.pediy.com/thread-210275.htm

脱完壳 dex文件，扔到 jeb 里面进行分析（GDA分析能力还是不太强，不过速度快）

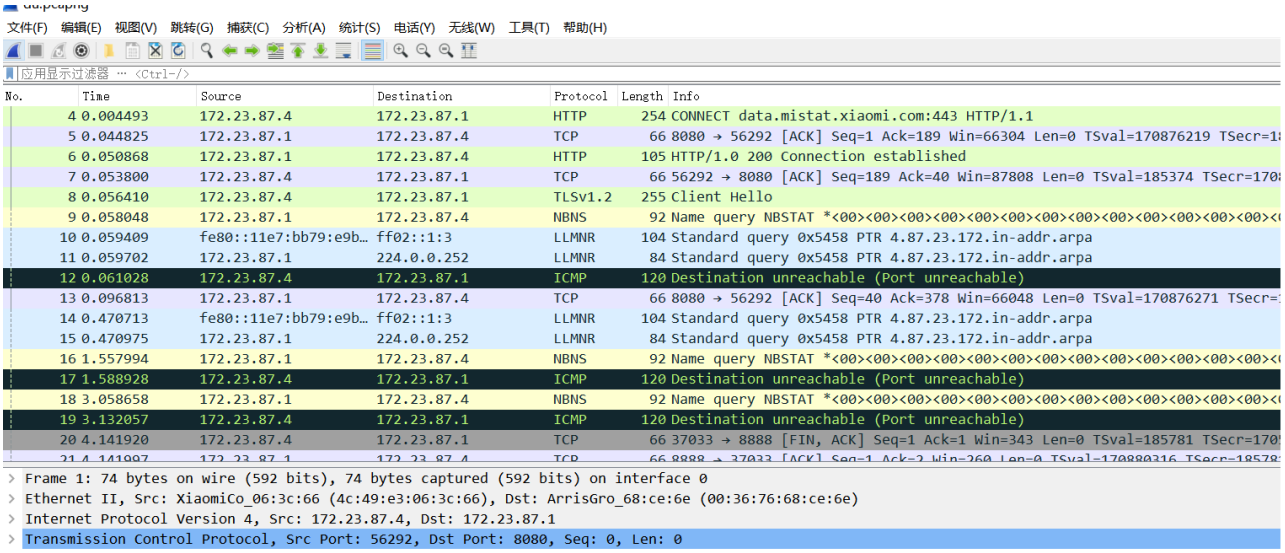
类和方法都出来了，脱壳成功。

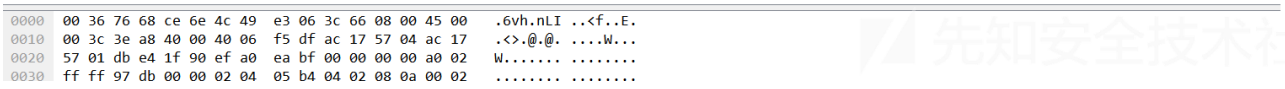
首先看看协议抓取，建议自己电脑起一个 ap （热点）， 然后用手机连接热点，对于 http 的数据包，可以使用 burp 进行抓取（对于 https 还要记得先安装 burp 的证书），对于 tcp 的数据包，由于我们是连接的 电脑的 wifi 所以我们可以直接用 wireshark 抓取我们网卡的数据包就能抓到手机的数据包。对于笔记本，可以买个无线网卡。

首先看看注册数据包的抓取，设置好代理，选择注册功能，然后去 burp 里面，可以看到抓取的数据包。

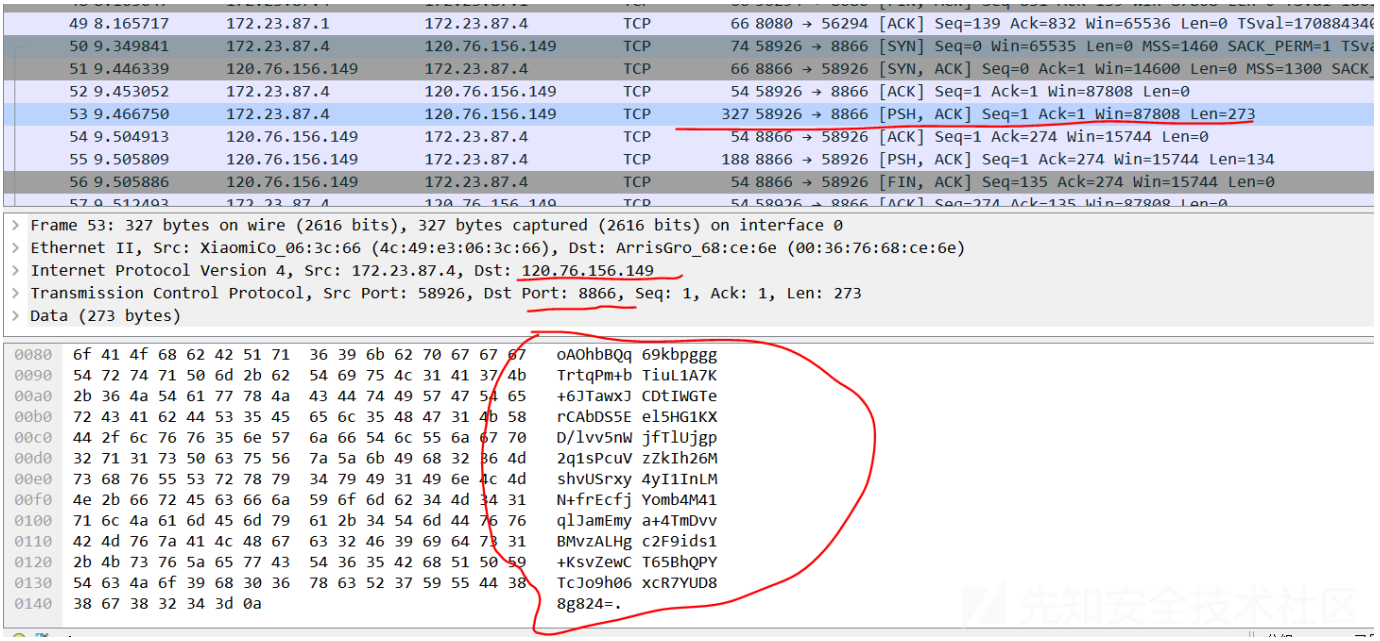


对于登录数据包，点击登录功能，去发现 burp 无法抓到数据包， 怀疑使用了 tcp 发送请求数据，于是开启 wireshark 抓取 手机连接的热点到的网卡的数据包。抓取时间要短一些，不然找信息就很麻烦了。





然后我们一个一个 `tcp` 数据包查看，看看有没有什么特殊的。

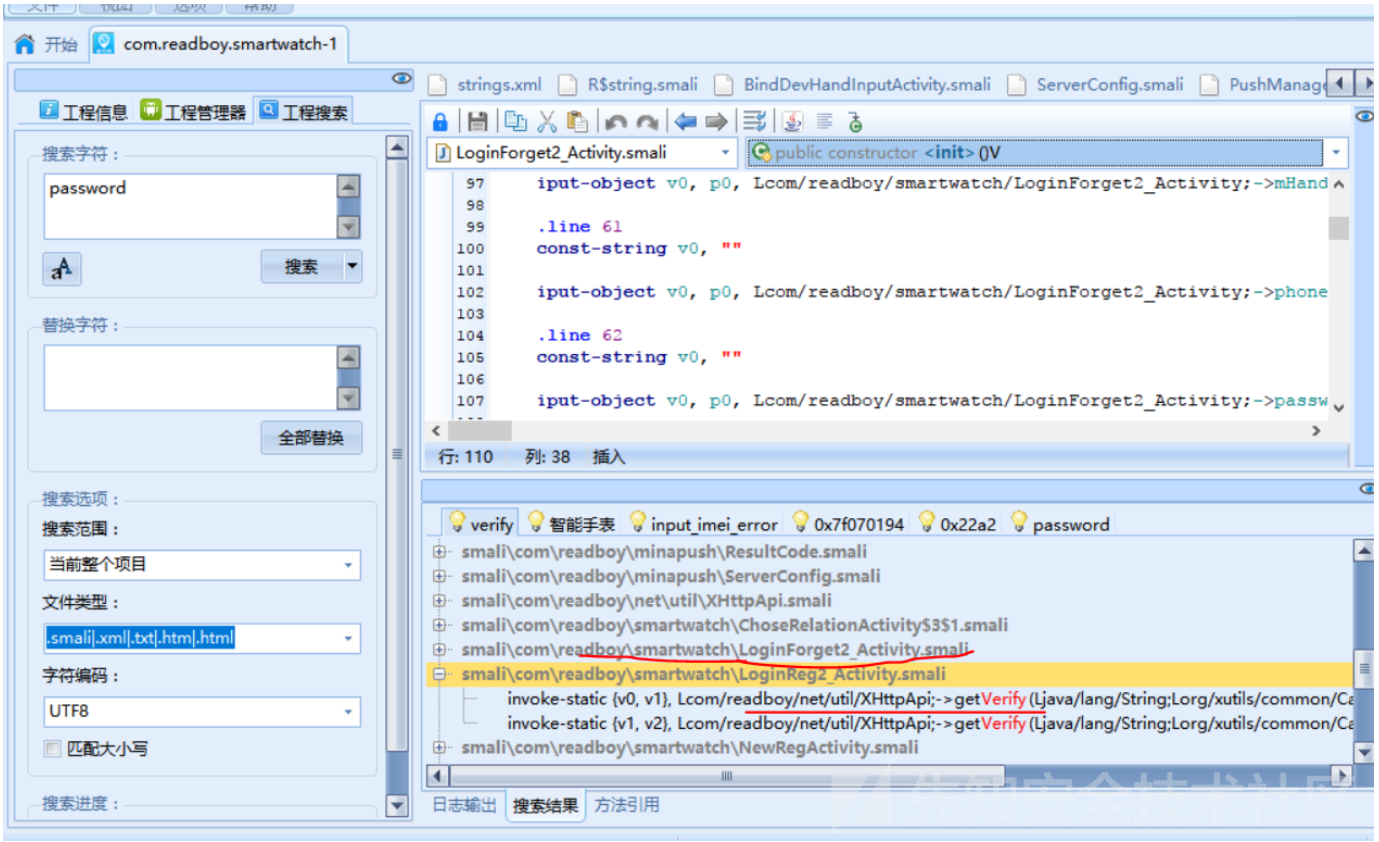


发现一个数据包里面有 `base64` 加密的数据，猜测这个应该就是登陆的数据包。查了一下 `ip` ，应该就是了。

下面针对不同类型的协议加密措施进行分析。

HTTP协议

协议分析关键是找到加密解密的函数，可以使用关键字搜索定位。为了方便搜索，先把 `dex` 转成 `smali` 然后用文本搜索工具搜索就行了，我使用 `android killer`。在这里可以使用 `sn` ， `verify` 等关键词进行搜索，定位关键代码。我选择了 `verify` ，因为它搜出的结果较少。



函数没经过混淆，看函数名就可以大概猜出了作用，找到关键方法，拿起 `jeb` 分析之。

先来看看 `LoginReg2_Activity` 的 `onCreate` 方法。

```
protected void onCreate(Bundle arg7) {
    super.onCreate(arg7);
    this.setContentView(2130968646);
    x.view().inject(((Activity)this));
    this.mToolbar = this.findViewById(2131624115);
    this.setSupportActionBar(this.mToolbar);
    this.titleView.setText(2131165825);
    Bundle v0 = this.getIntent().getExtras();
    if(v0 != null) {
        this.phone = v0.getString("$SwitchMap$com$github$mikephil$charting$charts$ScatterChart$ScatterShape");
    }

    this.mHandler.postDelayed(this.runnable, 1000);
    this.stateTv.setText(2131166039);
    XHttpApi.getVerify(this.phone, new sendCheckCodeCallBack(this, null));
    this.passwordEt.addTextChangedListener(this.editWatcher);
    this.passwordEt.setOnEditorActionListener(new TextView$OnEditorActionListener() {
        public boolean onEditorAction(TextView arg3, int arg4, KeyEvent arg5) {
            boolean v0;
            if(arg4 != 2) {
                if(arg5 != null && arg5.getKeyCode() == 66) {
                    goto label_6;
                }
```

获取手机号进入了 `XHttpApi.getVerify` 方法，跟进

```
public static Callback$Cancelable getVerify(String arg4, Callback$CommonCallback arg5) {
    MyLog.i("-------------------------getVerify1");
    RequestParams params = new RequestParams(ServerConfig.getMobileUrl() + "/verify");
    MyLog.i("-------------------------getVerify2");
    params.addQueryStringParameter("mobile", arg4);
    params = XHttpApi.addSnToParams(params);
    params.setCancelFast(true);
    return x.http().post(params, arg5);
}
```

先调用了 `XHttpApi.addSnToParams(params)` （看名称估计他就是增加签名的函数了），然后发送 `post` 请求。

继续跟进 `XHttpApi.addSnToParams`

```
private static RequestParams addSnToParams(RequestParams arg4) {
    arg4.addQueryStringParameter("appid", "AndroidWear");
    String time_stamp = String.valueOf(MyTimeUtils.getTimestamp());
    arg4.addQueryStringParameter("sn", StringUtil.getMd5("AndroidWear65cbcdeef24de25e5ed45338f06a1b37" + time_stamp));
    arg4.addQueryStringParameter("t", time_stamp);
    return arg4;
}
```

至此签名方案非常清晰了。

- 获取时间戳,新增一个 `t` 的参数，值为 时间戳
- `md5("AndroidWear65cbcdeef24de25e5ed45338f06a1b37" + time_stamp)` 为 `sn`

由于有时间戳和签名的存在，而且服务器会检测时间戳，后续如果我们想测试一些东西，就需要过一段时间就要计算一下 签名和时间戳，这样非常麻烦，我们可以使用 `burp` 编写插件，自动的修改 时间戳和签名，这样可以大大的减少我们的工作量。

看看关键的源代码

首先注册一个 `HttpListener`, 这样 `burp` 的流量就会经过我们的扩展。

```java
@Override
public void registerExtenderCallbacks(IBurpExtenderCallbacks callbacks) {

    callbacks.setExtensionName("monitor_http");

    this.hps = callbacks.getHelpers();
    this.cbs = callbacks;
    this.stdout = new PrintWriter(callbacks.getStdout(), true);

    this.stdout.println("hello burp!");
    this.cbs.registerHttpListener(this);

    SwingUtilities.invokeLater(new Runnable() {...
    });
}
```

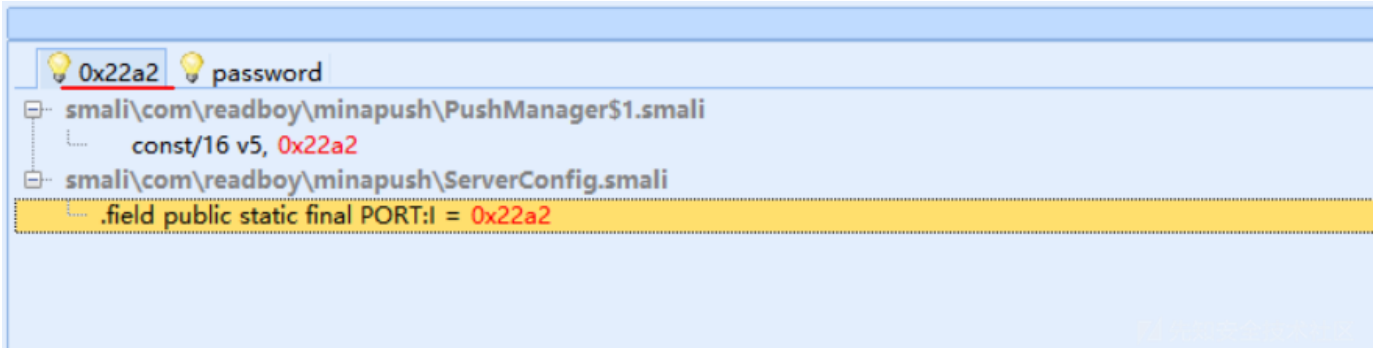

然后看看 `processHttpMessage`对流经扩展的流量进行处理的逻辑。只处理 `http` 请求的数据，然后根据域名过滤处理的数据包，只对 `wear.readboy.com` 进行处理。接着对于数据包中的 `t` 参数和 `sn` 参数进行重新计算，并且修改 数据包中的对应值。

```java
@Override
public void processHttpMessage(int toolFlag, boolean messageIsRequest, IHttpRequestResponse messageInfo) {

    try {
        if (messageIsRequest) {
            IRequestInfo analyzeReq = this.hps.analyzeRequest(messageInfo);
            new_Request = messageInfo.getRequest();
            this.stdout.println("host: " + analyzeReq.getUrl().getHost());
            if (analyzeReq.getUrl().getHost().indexOf("wear.readboy.com") > -1) {

                List<IParameter> paraList = analyzeReq.getParameters();
                String time_stamp = String.valueOf(Utils.getTimestamp() + 10000);
                for (IParameter para : paraList) {
                    this.stdout.println(para.getName());

                    String key = para.getName(); //获取参数的名称
                    String value = para.getValue(); //获取参数的值

                    this.stdout.println("is_t: " + key.equals("t"));
                    try {
                        if (key.equals("t")) {
                            IParameter newPara = this.hps.buildParameter(key, time_stamp, para.getType());

                            new_Request = this.hps.updateParameter(new_Request, newPara); //构造新的请求包
                            messageInfo.setRequest(new_Request);//设置最终新的请求包
                        } else if (key.equals("sn")) {
                            IParameter newPara = this.hps.buildParameter(key, Utils.getMd5("AndroidWear65cbcdeef24de25e5ed45338f06a1b37" + time_
                            new_Request = this.hps.updateParameter(new_Request, newPara); //构造新的请求包
                            messageInfo.setRequest(new_Request);//设置最终新的请求包
                            this.stdout.println("new time_stamp:  " + time_stamp + " sn:" + newPara.getValue());
                        }


                    } catch (Exception e) {
                        e.printStackTrace();
                    }

                }
```


加载扩展，以后重放数据包，就不用管签名的问题了。

TCP

对于 `tcp` 的协议可以通过搜索 端口号，`ip` 地址等进行定位，这里搜索 `端口号`（这里是8866, 可以在 `wireshark` 中查看），有一点要注意，程序中可能会用 `16` 进制或者 `10` 进制表示端口号为了，保险起见建议两种表示方式都搜一下。



通过搜索 0x22a2 （8866 的 16 进制表示）找到两个可能的位置。分别检查发现 第二个没啥用，在 jeb 中查找交叉引用都没有，于是忽略之。然后看看第一个。

```
FutureTask v1 = new FutureTask(new Callable() {
    public Boolean call() {  // 建立tcp连接
        try {
            PushManager.this.connectFuture = PushManager.this.connector.connect(new InetSocketAddress(ServerConfig.getHostName(), 8866));
            PushManager.this.connectFuture.awaitUninterruptibly();
            PushManager.this.session = PushManager.this.connectFuture.getSession();
            PushManager.this.session.getConfig().setWriteTimeout(5000);
            MyLog.i("--------connect sessionId = " + PushManager.this.session.getId());
        }
        catch(Exception v0) {
            v0.printStackTrace();
            Boolean v1 = Boolean.valueOf(false);
            return v1;
        }

        return Boolean.valueOf(true);
```

可以看到 jeb 把端口号都转成了 10 进制数，这里与服务器进行了连接，没有什么有用的信息。于是上下翻翻了这个类里面的函数发现一个有意思的函数。

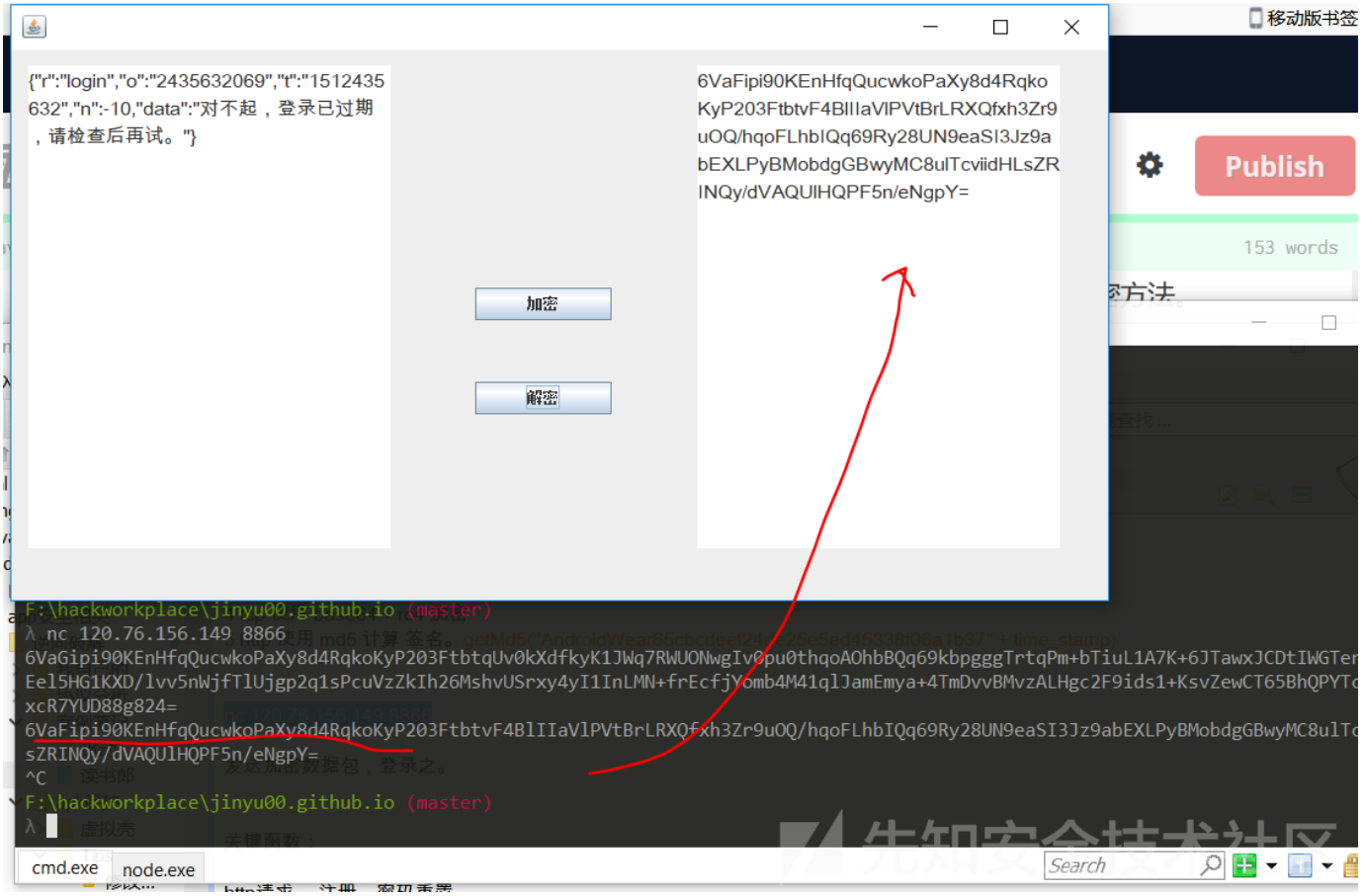
```
public boolean sendMessage(String arg9) {
    boolean v1 = false;
    if(!this.isConnected()) {
        MyLog.e("----------app is not connect server");
        return v1;
    }

    MyLog.i("------time = " + MyTimeUtils.timestampToDateString(Long.valueOf(System.currentTimeMillis() / 1000), "-->start download activity exce
    try {
        arg9 = MyTimeUtils.data_encrypt(arg9);
    }
    catch(Exception v0) {
        v0.printStackTrace();
        return v1;
    }

    this.session.write(arg9);
    return true;
}
```

用于发送数据，里面还用了另外一个类的方法，一个一个看，找到了加密方法。

```
public static String data_encrypt(String raw_data) throws Exception {
    Cipher cipher = Cipher.getInstance("RC4");
    cipher.init(1, new SecretKeySpec("22690dfba7ab83b4".getBytes("UTF-8"), "RC4"));
    return new String(Base64.encode(cipher.update(raw_data.getBytes(Charset.forName("UTF-8"))), 2), Charset.forName("UTF-8"));
}
```

就是简单的 rc4 加密，然后在 base64 编码。
为了测试的方便写了个图形化的解密软件。

用 nc 测试之



正确。

总结

不要怕麻烦，一些东西尽早脚本化，自动化，减轻工作量。逆向分析，搜索关键字，定位关键代码。

参考

http://www.vuln.cn/6100

http://www.freebuf.com/articles/terminal/106673.html

来源： https://www.cnblogs.com/hac425/p/9416890.html