登录

Android逆向工具：非通常方法

Edvison / 2018-07-25 10:17:01 / 浏览数 4861 技术文章 技术文章 顶(1) 踩(0)

本文是Android reverse engineering tools: not the usual suspects的翻译文章。

摘要

在Android安全领域，所有逆向工程师可能都会使用一些著名的分析工具，如apktool，smali，baksmali，dex2jar等。这些工具确实是Android应用程序分析的必备工具。但是，还有其他有趣的工具和问题，很少在会议中涉及，这些都是本文的内容。

本文讨论下面五个点：（1）如何为Android应用程序分析共享逆向工程环境，（2）如何编写JEB2脚本，（3）Android调试器的状态，（4）如何读取TLS加密通信，（5）如何在Dalvik上使用Radare2。

1.简介

Android逆向是现在在互联网上广泛涉及的主题，这里有一些关于Android逆向的视频，文章教程[1, 2, 3, 4]，所有这些都提供了大致相同的步骤（什么是APK，什么是Android清单，如何解压缩，阅读smali，反编译，重新组装APK）并使用相同的工具（apktool，smali / baksmali，AXMLPrinter，Java Decompiler，Androguard）。 事实上，这种方法是完美的，工具非常方便，我鼓励阅读这些链接。

那些在该领域工作的人（例如反病毒分析员）也面临许多其他高级问题，其中之一是需要对高级样本进行解包或去混淆。 研究人员已经多次讨论过这种情况[5, 6, 7, 8]。但是，其他几个问题仍然没有答案。 在本文中，我们将解决以下问题：

- 如何与他人共享Android程序的逆向环境。
- 如何编写Android逆向工具脚本以简化样本分析。
- 是否可以调试可疑应用程序，设置断点并逐步运行它以了解发生的情况。
- 随着Android程序开发人员现在越来越多地使用HTTPS，我们将考虑如何检查SSL / TLS加密的应用程序流。
- 我们是否可以超越常规，使用除了通常的apktool，baksmali之外的反汇编程序。
  本文从移动反病毒分析师的角度处理这些独立问题。 但是，这些技术适用于任何Android程序的逆向。 每个问题都在自己的章节中讨论。

2. 适用于Android逆向的Docker镜像

2.1 为什么使用Docker镜像？

反病毒分析师必须在干净的环境中检查每个可疑样本。
有时，您还需要与同事共享样本，以获得他/她对特定点的建议。唉，建立一个逆向工程环境非常耗时。严格来说，这并不困难，但有许多不同的工具可以安装（apktool，b
Decompiler，AXMLPrinter，Android模拟器...），每个都有不同的设置步骤，没有自动化程序。

因此，Android程序的便携式逆向环境是最有用的。
为了创建这样的环境，一些计划已经提出了具有所有必要工具的虚拟机。例如，有Android逆向工程（ARE）VM [9]或Androl4b VM [10]。
VM的缺点是您必须下载千兆字节的信息（整个Linux主机以及Android工具），并且您获得的环境通常已经过时，因为VM尚未维护。

为了解决这些问题，我提出了使用Android逆向的Docker镜像。 Docker [11]是一个开源项目，可以自动在软件容器内部署应用程序[12)]。
与完整VM相比，下载大小减少，因为：

- Docker容器依赖于主机的底层操作系统，不需要包含完整的操作系统[13]。
- Docker镜像由多个层组成，就像下载块一样。
  这些图层可以在镜像中重复使用，因此如果给定镜像使用图层A和B，而另一镜像使用B和C，则不需要重新下载B.

我的Docker镜像已经上传到Docker Hub了，可以使用Docker命令docker pull直接下载。 确切地说，使用我的环境的步骤是：

- 安装Docker（如果你还没有这样做的话）。
- 检索Android RE Docker镜像：
  `docker pull cryptax/android-re`
- 启动一个（或多个）容器：
  `docker run -d options cryptax/android-re`
  您可以立即获得一个独立的Linux主机，您可以通过该主机SSH或使用VNC（桌面共享）并访问Android逆向工具和模拟器。
  Docker镜像的使用也改善了维护问题。 实际上，Docker镜像是从Dockerfile构建的，就像创建镜像的"源脚本"一样。
  例如，Dockerfile指定要安装的每个包以及设置容器所需的命令。
  这个Dockerfile在[14]共享，你可以根据自己的需要自由定制，或者将它更新到这个或那个更新的版本。

2.2 镜像创建技巧

在本小节中，我分享了一些技巧，用于为Docker镜像设置Dockerfile。

2.2.1 Docker 和 GUI

Docker特别适合创建包含守护进程，服务，Web服务器等的隔离专区。令人惊讶的是，对图形应用程序的支持并不容易，并且有许多问题和博客文章[15]。
基本上，Docker容器被视为远程Unix主机，有三种选择（可见下表）：

- 与容器共享您的显示器。 运行xhost +。 然后启动容器，并在选项中确保共享DISPLAY环境变量和X11套接字：

```
docker run -d -e DISPLAY=$DISPLAY -v /tmp/.X11-unix:/tmp/.X11-unix
OTHER-OPTIONS --name mycontainer cryptax/android-re
```

- 在SSH中使用X转发。 这再次依赖于X11。 通过SSH连接到容器时，请指定选项-X以启用X11转发。
  使用VNC。 在这种情况下，图形应用程序的显示由容器处理。 容器包含VNC服务器（图形桌面共享系统）。
  使用容器时，将VNC客户端连接到容器的VNC服务器并共享其桌面。
  Android模拟器需要GLX支持。 因此，容器必须配置支持GLX的X11显示服务器。 例如：

```
Xvfb :1 +extension GLX +render -noreset -screen 0 1280x1024x24&
DISPLAY=:1 /usr/bin/xfce4-session >> /root/xsession.log 2>&1 &
```

解决方法	优点	缺点
分享显示	轻量级	仅适用于支持X11的主机（有时会有错误）
SSH X转发	轻量级	不适用与Windows,Mac上有bug
VNC查看器	需要安装vncviewer	由容器设置的屏幕分辨率。容器窗口出现在VNC查看器窗口内。 容器需要包含窗口管理器和X11服务器。

2.2.2 两个命令

因此，如果我们想要更灵活地使用，我们的Docker镜像将需要（1）SSH服务器和（2）容器中的VNC服务器。 通常，服务器通过CMD在Dockerfile中启动。
例如：
`CMD ["/usr/sbin/sshd", "-D"]`
如果我们为VNC指定另一个CMD，我们会感到惊讶：Dockerfiles不支持多个CMD。 最后一个CMD取代之前的所有CMD。 推荐的解决方案是使用supervisor[16]，一个过程控制系统。
在我们的例子中，我们配置supervisor启动SSH和VNC，并运行supervisord。 在下图中，第4行和第5行配置SSH服务器。
第6行和第7行启动个人脚本startXvfb.sh，它启动Xvfb和VNC服务器。

```
# Configure supervisor                                          1
RUN echo "[supervisord]" >>                                     2
  ↪   /etc/supervisor/conf.d/supervisord.conf
RUN echo "nodaemon=true" >>                                    3
  ↪   /etc/supervisor/conf.d/supervisord.conf
RUN echo "[program:sshd]" >>                                   4
  ↪   /etc/supervisor/conf.d/supervisord.conf
RUN echo "command=/usr/sbin/sshd -D" >>                        5
  ↪   /etc/supervisor/conf.d/supervisord.conf
RUN echo "[program:startxvfb]" >>                              6
  ↪   /etc/supervisor/conf.d/supervisord.conf
RUN echo "command=/bin/sh                                      7
  ↪   /root/startXvfb.sh" >>
  ↪   /etc/supervisor/conf.d/supervisord.conf
...                                                            8
CMD [ "/usr/bin/supervisord" ]                                 9
```

2.2.3 密码

登录VNC服务器的密码在Dockerfile中进行了硬编码：

```
ENV VNC_PASSWORD "rootpass"
RUN x11vnc -storepasswd $VNC_PASSWORD ~/.vnc/passwd
```

SSH密码也是硬编码的。 由于我们需要一个简单的环境，因此只有一个用户root。 所以root需要能够SSH。 默认情况下这是不可能的;
需要修改SSH服务器配置以授权root登录：

```
RUN echo "root:$SSH_PASSWORD" | chpasswd
RUN sed -i 's/PermitRootLogin prohibit-password/PermitRootLogin yes/' /etc/ssh/sshd_config
RUN sed 's@session\s*required\s*pam_loginuid.so@session optional pam_loginuid.so@g' -i /etc/pam.d/sshd
```

请注意，您可能希望针对自己的环境进行更改。 使用硬编码密码也意味着Docker容器不应在Web上公开提供。

2.2.4 Android模拟器

Dockerfile中的一个重要步骤是安装Android模拟器。 通常，Android模拟器使用名为android的图形工具安装。
实际上，android也可以在命令行中运行，因此可以在Dockerfile中使用。 步骤是：

下载Android SDK，解压缩并设置路径。

```
RUN wget -q -O "/opt/tools-linux.zip" https://dl.google.com/android/repository/...
RUN unzip /opt/tools-linux.zip ...
...
ENV PATH $PATH:/opt:...
```

然后，您可以使用android更新SDK工具，平台工具和构建工具，获取Android版本并获取给定体系结构（例如ARM）的系统映像：

```
RUN echo y | android update sdk --filter tools --no-ui --force -a
RUN echo y | android update sdk --filter platform-tools --no-ui --force -a
```

由于命令行android要求用户确认，因此需要echo y以在用户输入的后台运行命令。

创建Android虚拟设备（AVD）

```
RUN echo n | android create avd --force --name AVDNAME --target ANDROID-VERSION --abi "default/armeabi-v7a"
```

最后一步是导出Android模拟器使用的端口。 默认情况下使用的第一个控制台端口是5554.
Docker容器必须打开该端口，以便您可以远程登录到Android模拟器控制台。

2.2.5 磁盘空间

最后，要考虑磁盘空间。 如果我们在Dockerfile中配置了许多软件包，那么容器将是巨大的，与使用VM相比，我们将获得很少的收益。
Dockerfile只需要与使用的东西一起设置（你可以自定义我的，删除你不需要的东西）。 此外，清理包缓存也很好。
网上有几个最佳实践，解释了如何优化一个Dockerfile [17]。

3. JEB2脚本

JEB是一个图形化的Android程序反编译器，由PNF Software商业化，并且经常被该领域的人们使用。 与IDA
Pro类似，逆向工程任务可以通过Python代码自2.0.14版（JEB2）编写脚本。更有趣的是自动执行必须针对给定分析执行的可重复的繁琐任务。
PNF软件提供文档[18]和博客文章[19]来帮助编写您的第一个脚本，但这些示例都过于简单（打印hello world）并不能帮助逆向工程师完成实际任务。
本节将介绍如何实现字符串反混淆器 - 反转恶意样本时的常见要求 - 作为JEB2脚本，以Android /
Ztorg示例为例（sha256：2c546ad7f102f2f345f30f556b8d8162bd365a7f1a52967fce906d46a2b0dac4）。

3.1 为脚本设置JEB2

第一个预备步骤是安装。 在文档中并不总是那么清楚，幸运的是这很简单：首先安装Jython（Python for
Java平台），然后将脚本放在./JEB-HOME/scripts目录中。 此外，请确保为API参考[20]添加书签，因为这是脚本开发的必备条件。

3.2 去混淆脚本目标

我们注意到Ztorg示例使用字符串混淆。 混淆的字符串作为字节数组加载（没有明显的意义）并由例程解码 -
特别是在这种情况下，通过包a.b中的类c的方法a()（样本也模糊了名称，如您所见） - 解码例程已在[21]中反转。

```
v0[1] = c.a(new byte[]{83, 115, 47, 37, 47, 115, 49, 51, 56, 41, 48, 57, 115, 42, 62, 51, 36, 59, 41, 57, 47, 40, 115, 47, 57, 6
v0[2] = c.a(new byte[]{118, 120, 36, 46, 36, 120, 58, 56, 51, 34, 59, 50, 120, 33, 53, 56, 47, 33, 62, 51, 50, 56, 120, 62, 57,
v0[3] = c.a(new byte[]{106, 40, 116, 126, 116, 40, 106, 104, 99, 114, 107, 98, 40, 113, 101, 104, 127, 96, 114, 98, 116, 115, 40
v0[4] = c.a(new byte[]{116, 27, 71, 77, 71, 27, 89, 91, 80, 65, 88, 81, 27, 66, 86, 91, 76, 66, 93, 80, 81, 91, 27, 92, 91, 88,
v0[5] = c.a(new byte[]{48, 109, 49, 59, 49, 54, 39, 47, 109, 39, 54, 33, 109, 43, 44, 43, 54, 108, 32, 55, 43, 46, 38, 48, 45, 4
v0[6] = c.a(new byte[]{23, 15, 68, 69, 86, 15, 86, 66, 79, 88, 85, 83, 69, 82, 55});
v0[7] = c.a(new byte[]{102, 85, 9, 3, 9, 85, 23, 21, 30, 15, 22, 31, 85, 12, 24, 21, 2, 29, 15, 31, 9, 14, 85, 10, 27, 8, 27, 23
v0[8] = c.a(new byte[]{101, 84, 8, 2, 8, 84, 31, 30, 13, 18, 24, 30, 8, 84, 13, 18, 9, 15, 14, 26, 23, 84, 22, 18, 8, 24, 84, 13
```

我们希望找到初始化这些混淆字符串的所有类，并自动对它们进行去混淆。 结果如图所示。

```
v0[1] = "/sys/module/vboxguest/sections/.strtab";
v0[2] = "/sys/module/vboxvideo/initsize";
v0[3] = "/sys/module/vboxguest/sections/.symtab";
v0[4] = "/sys/module/vboxvideo/holders";
v0[5] = "/system/etc/init.buildroid.sh";
v0[6] = "/dev/vboxuser";
v0[7] = "/sys/module/vboxguest/parameters";
v0[8] = "/sys/devices/virtual/misc/vboxuser/subsystem";
```

3.3 脚本开发

我们提到官方文档太基础了，但是PNF软件在GitHub上提供了几个示例脚本[22]。 最接近我们需求的是JEB2JavaASTDecryptStrings.py，它是我们脚本的基础。
我的脚本可在[23]获得。
基本上，我们保留示例的开头：导入，后端引擎的初始化，打开第一个项目并枚举反编译的类。
原始脚本（类重命名，删除了一些不必要的行）的变化很少，直到那一点（JEB2JavaASTDecryptStrings.py的第45行 - 对应于我自己脚本的第35行）。

```
self.units = RuntimeProjectUtil.findUnitsByType(prj, IJavaSourceUnit, False)
```

在Ztorg示例中，我们注意到反编译的字符串总是位于静态类构造函数中：

```
static {
b.a = c.a(new byte[]{15, 116, 8});
b.b = c.a(new byte[]{110, 114, 105, 111});
b.c = c.a(new byte[]{105, 4, 25, 8, 21,
,! 107, 8});
b.d = c.a(new byte[]{85, 29, 66});
}
```

所以，第一步是找到静态构造函数：

- 1.获取代表该类的JEB2对象：

  ```
  javaClass = unit.getClassElement()
  ```

  在API中，这将返回IJavaClass类型的对象（请参阅API中的IJavaSourceUnit中的getClassElement）。
- 解析类的所有方法（getMethods（））。
- 检查类的名称是否对应于静态构造函数。 之后我们可以休息。

  ```
  if m.getName() == '<clinit>'
  ```

  然后，我们需要在静态构造函数中找到涉及对去混淆例程的调用的所有行。 在JEB2中，行更精确。 我们解析方法的语句：

  ```
  for statement in m.getBody():
  ```

  有几种类型的语句：函数调用，赋值，条件，返回等。在我们的示例中，模糊字符串出现在赋值中。 所以，我们过滤赋值语句：

  ```
  if statement.getElementType() == JavaElementType.Assignment:
  ```

  这是一个有点棘手的地方，混淆字符串出现在：
- 简单的任务。

  ```
  v = c.a(....)
  ```

  我们使用诸如statement.getRight（）之类的调用检索的赋值的右侧就是对去混淆例程的调用。 这就是我们需要修改的内容

  ```
  v = ''de-obfuscated''
  ```

- 更复杂的任务

  ```
  v = new String(c.a(....))
  ```

  赋值的右侧不是调用而是new，它包含对去混淆例程的调用。 我们希望将其转换为：

  ```
  v = new String(''de-obfuscated'')
  ```

  因此，要检查语句是否调用去混淆例程，我们必须：
- 检查语句的getRight（）是否是对我们例程的调用（例1）。 我们通过检查它的签名La / b / c来匹配例程; - > a（[B]Ljava/lang/String;（记住，解码例程是a.b.c.a（））：

```
getMethod().getSignature()
```

否则，检查右侧部分是否包含对我们的例程调用的子元素（案例2）。 我们解析元素:

```
for rightsub in statement.getRight().getSubElements():
```

当我们找到这样的声明时，就需要去混淆。 这就是称之为去混淆方法的点。
最后，我们可以在控制台中打印结果，但更好的是，我们希望去混淆的字符串用c.a（...）替换该部分。
下面代码的第1行调用replaceSubElement（）完成这个操作，其中：

- elem是包含c.a（...）的右侧部分，例如statement.getRight（）。
  father是包含该元素的元素。 例如，上层的右侧部分或语句。
  使用self.cstbuilder.createString（）（第1行）创建新的去混淆字符串，并通过通知它来更新JEB2窗口（unit.notifyListeners（） - 第2行）。

```
father.replaceSubElement(elem, self.cstbuilder.createString(''.join(map(chr,decbytes))))
unit.notifyListeners(JebEvent(J.UnitChange))
```

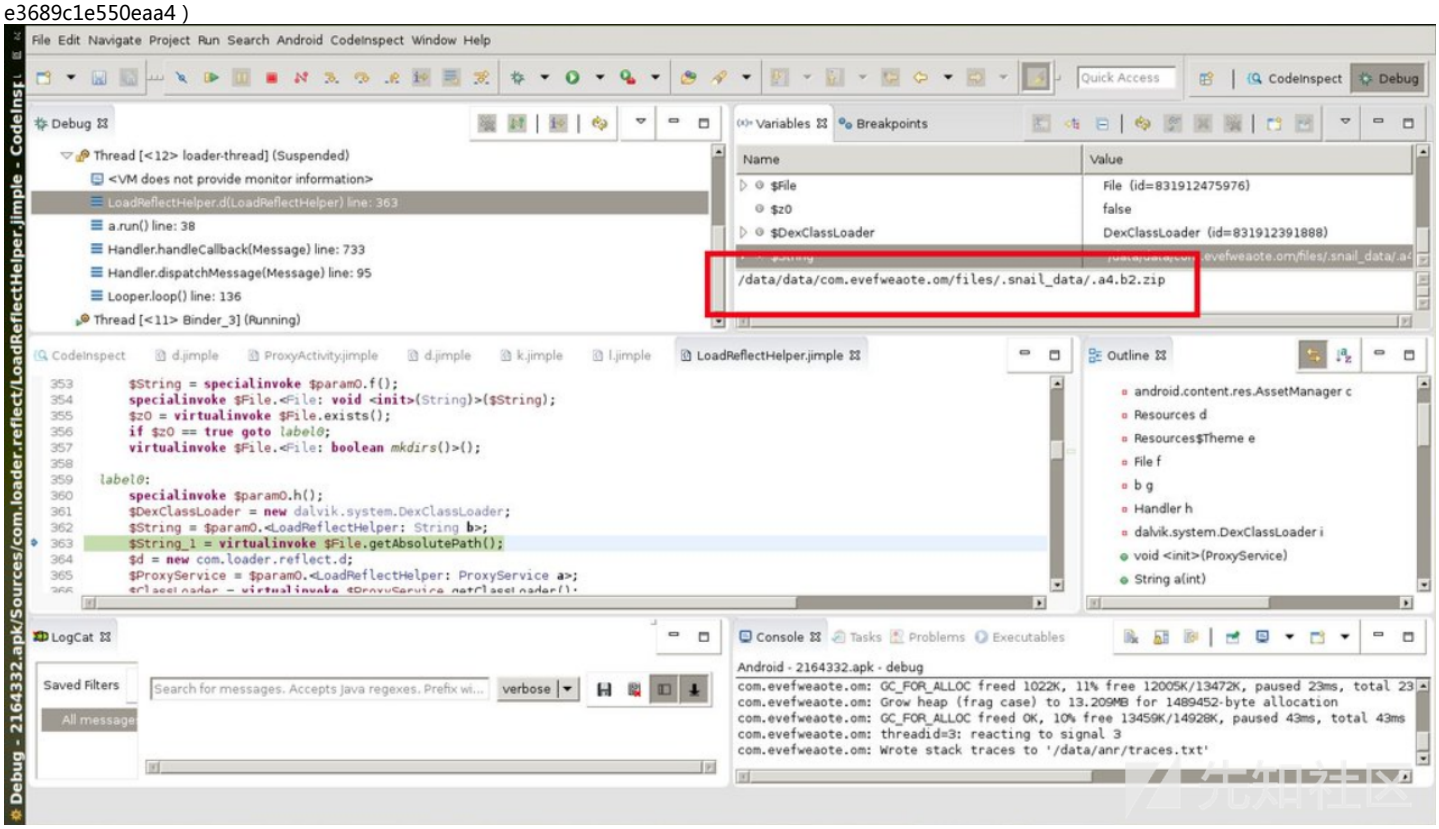
4. 调试

Android样本调试是许多逆向工程师的梦想。 特别是在复杂的样本上，将断点放在关键线上，逐步运行代码，检查（甚至修改）变量和堆栈非常方便。
据我所知，有两种工具可以在Dalvik级别完成：JEB2（我们在第3节中提到）和CodeInspect [24]。

我尝试了JEB2 2.2.11版本（撰写本文时的最新版本）和CodeInspect（2016年10月的许可演示版）。 结果对未来有希望，但尚未成熟。

4.1 CodeInspect

我在CodeInspect面临的主要问题是它的重量级。 我花了将近三分钟来打开一个调试器。
不过，如果你足够耐心，它运行良好，我成功调试了实例Riskware/InnerSnail!Android (sha256: c5c11408483eb87781af30280b2878890f5741fe63d569ae9e3689c1e550eaa4）



该示例使用DexClassLoader类加载Dalvik可执行文件。 该文件作为类构造函数中的参数传递，但是使用静态分析我无法找到它的值。
所以我在CodeInspect中打开了这个项目。 Dalvik字节码被转换为Jimple，这是Java的中间表示。 它与smali不同，但易于遵循。
我在相应的行上设置断点，打开调试器并将其附加到现有的模拟器（或者，CodeInspect可以启动另一个）。
它将示例安装在模拟器上，运行到断点并读取变量的值（隐藏的zip文件名）。

4.2 JEB2

使用JEB2，步骤基本相同，除了JEB2 GUI不安装和运行应用程序 - 您需要这样做。 要启动应用程序：

```
am start -D -S -n PACKAGENAME/ACTIVITYNAME
```

其中包名称类似于com.mx.cool.videoplayer，活动名称是包名称的相对路径，例如， .activity.MainActivity（不要忘记初始点）。

我尝试了两个不同的样本：Android/Crosate.A!tr (sha256: 15281dbe2603f5973d53c5fddabbcc3de6ad3ec65146aa2ffb34a779ea604f82)和第3节的Ztorg。
不幸的是，我遇到了许多错误和崩溃（我向开发人员报告）当前版本的JEB2，很难完成工作。

4.3 结论

希望在未来几个月内，CodeInspect和JEB2的情况都会有所改善。
请注意，运行调试器会话显然会运行示例，因此，在发生严重恶意活动后，不要将断点设置得太远。
此外，如果您修改代码，它会重新编译一个新的应用程序，这可能会在恶意软件分析的情况下引发道德问题，因为它实际上会创建一个新的恶意样本。

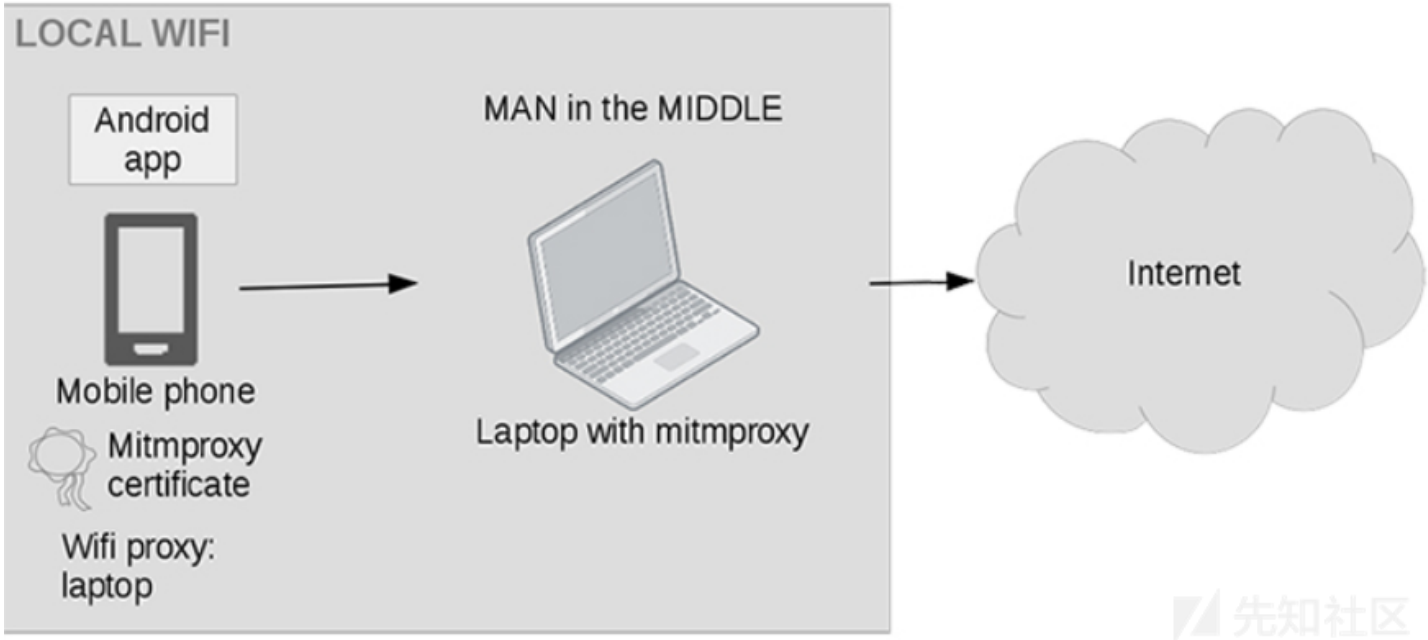
5. HTTPS 流量检查

好消息：越来越多的Android应用程序使用TLS与远程服务器通信。
然而，对于逆向工程师，尤其是反病毒分析师，这提出了另一个问题，因为通信流现在是加密的，因此是不可读的。 我们怎样才能解密流量？

解决方案是Man-in-the-Middle（MitM），我们拥有的主机配置为模拟客户端的服务器，相反，模拟服务器的客户端。
当客户端与服务器通信时，MitM主机拦截请求并提供自己的证书，声称它是服务器。 客户端被欺骗，因此加密MitM主机的消息，而不是服务器的消息。
服务器对客户端的响应以相同的方式处理，MitM主机此次声称它是客户端。

Mitmproxy能够自动执行此操作。 此工具在MitM主机上运行。
它为每次与TLS服务器的通信动态自动生成证书，并解密并显示流经它的数据包（甚至可以进行数据包修改）。

下图解释了我们逆向工程实验的架构。 Android智能手机和MitM主机位于同一个（Wi-Fi）网络上。



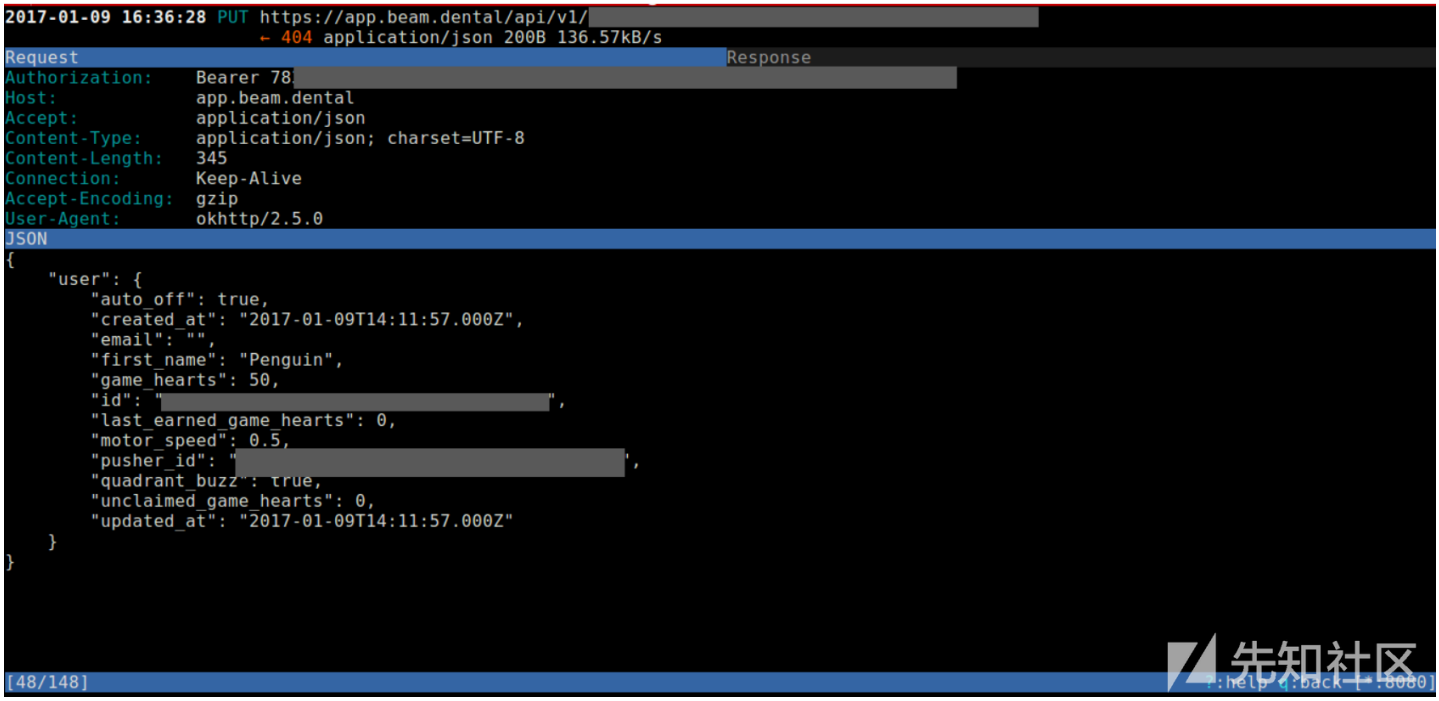
为了拦截网络数据包，我们修改手机的Wi-Fi连接配置以使用代理：指定MitM主机的IP地址，默认端口8080（还有其他可能性，但这是最简单的 - 见[25]）。
因此，手机的所有数据包都将转至MitM主机。
为了模拟真实服务器，MitM主机生成（假）服务器证书，由其自己的CA签名。 由于手机不知道此CA，因此必须将其添加到手机的SD卡中：

```
push /.mitmproxy/mitmproxy-ca-cert.cer
```

然后在手机上安装证书：设置 - >安全 - >从SD卡安装，然后选择证书。

设置完成。 启动mitmproxy以开始窃听手机和远程TLS服务器之间的通信。

例如，我在真正的Android应用程序上执行了这样的MitM来控制智能牙刷（图5）。 与远程服务器https://app.beam.dental
的通信是通过HTTPS进行的，并且在标准网络捕获中显示为加密。 通过mitmproxy，我们能够解密任何数据包并检查其内容。



请注意，存在一个限制：MitM不适用于使用证书锁定的Android应用。 但是，到目前为止，这种应用还很少见。

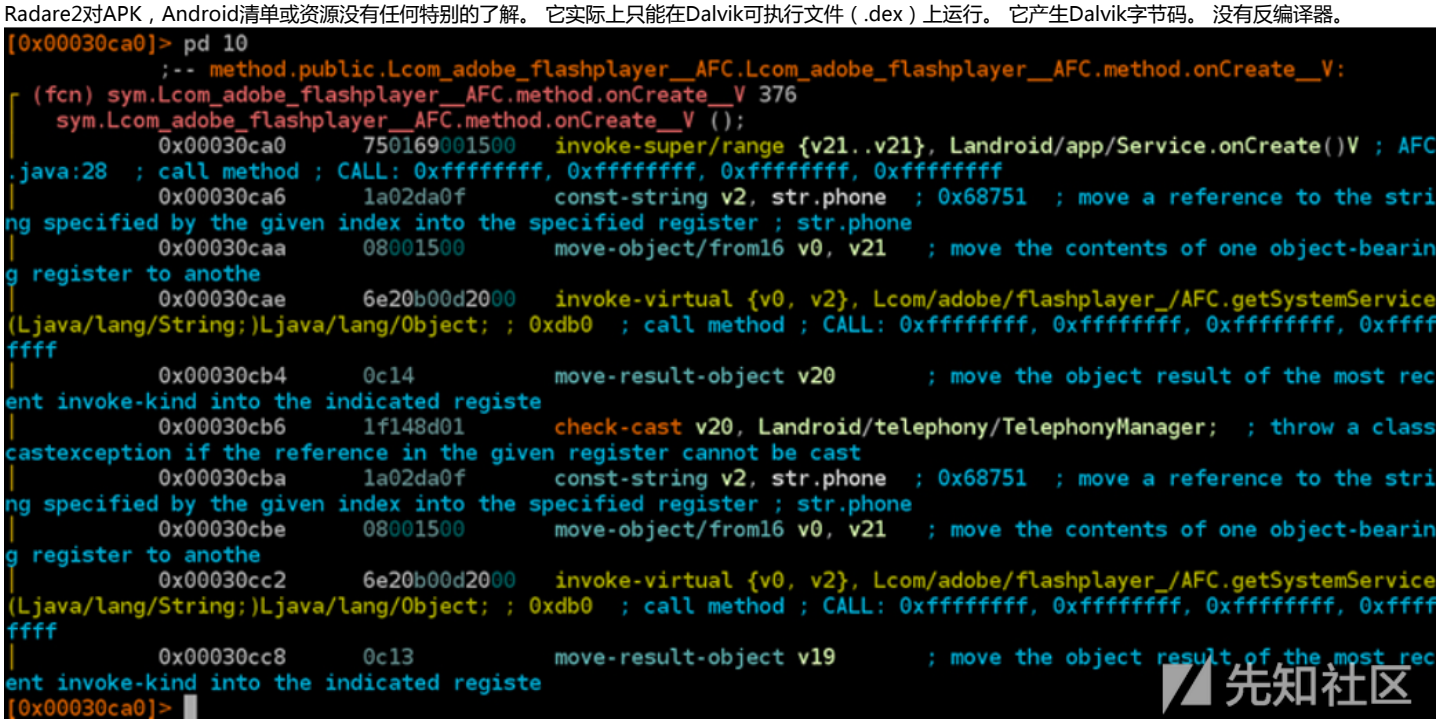
6. Radare2

Radare2是一个"逆向工程和分析二进制文件的框架"[26]。
它是开源的，并且在极客社区中因其命令行交互式shell而广为人知，并且广泛支持许多架构，包括较少使用的架构。

本节提供有关使用Radare2分析Android恶意软件的提示和技巧，以及反馈。

6.1 Dalvik支持

虽然Radare2不是逆向Android程序的直接选择（该领域的人通常更喜欢apktool，baksmali，JD，JEB等组合），但它最近增加了对Dalvik可执行文件的支持（实际上，对
Radare2对APK，Android清单或资源没有任何特别的了解。 它实际上只能在Dalvik可执行文件（.dex）上运行。 它产生Dalvik字节码。 没有反编译器。



反汇编程序相当不错，偶尔会有错误。 例如，在4月份，我报告了数组数据有效载荷的反汇编中的一个错误[27]。 这几天后就解决了。

6.2 逆向DEX的命令

有几个关于如何使用Radare2的教程（例如参见[28, 29, 30, 31]）。 在本小节中，我们只关注处理Dalvik可执行文件的特殊性。

首先，我们在DEX上启动Radare2：

```
r2 -e asm.payloads=true classes.dex
```

然后，我们需要用命令aa分析所有标志。 不幸的是，这个步骤目前在一些样品上很长（在某些情况下长达10分钟！）。

以下命令对DEX最有用：

搜索对DEX特别有用，因为可执行文件格式包含字符串池，但也包含类和方法的文本名称。
因此，grep对于给定的常量（iz~string），导入（ii~string），类名（ic~string），函数名（afl~string）和字节码（pd LINES @ FUNC~string）非常有用。 注意~和要搜索的字符串之间没有空格。 搜索区分大小写。 有些字符表现不佳，例如 斜线显示为下划线。

```
0x00064fef 49 str.http:__verisign_controlcenter.com_teapot_gate.php
```

方法分析。 您可以使用seek命令的ADDR或sf FUNC-SYMBOL-NAME跳转到给定方法，或直接在给定地址处反汇编几行：pd LINES @ ADDR。
查找交叉引用也是必须的：axt NAME用于交叉引用给定名称，axf NAME用于交叉引用来自给定名称。

```
[0x0001f424]> s 0x00036b30
or
[0x0001f424]> sf sym.Lcom_adobe_flashplayer__UC.method._init___V
[0x00036b30]> pd 10
...
[0x00036b30]> axf sym.Lcom_adobe_flashplayer__UC.method._init___V
C 0x36b30 invoke-direct {v0}, Ljava/lang/Object.<init>()V ; 0xede
```

注释/编辑。 要添加注释，命令是CC■■■■■■@ADDR。 要删除它，CC-。 要重命名一个函数：afn new-name。 目前尚无法重命名局部变量。 afvn v20新名称最终将在一天内推出。

6.3 脚本

由于其命令行性质，Radare2特别适合脚本编写。 例如，从Radare2提示符，可以使用以下特殊构造调用Python r2脚本：

```
#!pipe python ...
```

脚本本身必须导入r2pipe库[32]：

```
import r2pipe
```

并且您可以自动执行两个命令：

```
r2p=r2pipe.open()
r2p.cmd(your r2 command)
```

我编写了一个Radare2脚本来解混代Android / Ztorg示例的字符串。 该脚本可在[33]获得。 它有两个参数：混淆字符串的地址及其长度。 它将：

- 读取模糊字符串的地址（作为第一个参数提供）
- 跳转到该地址（命令s ADDR）
- 将x后面的字节（第二个参数）作为unicode字符串读取（命令p8 BYTES）
- 在这些字节上调用去混淆例程并显示结果。
  据我所知，这是Android恶意软件上Radare2的最先进用法。

6.4 讨论

本小节介绍了我对Radare2的个人印象。

我在简单和复杂的样本上都使用了Radare2：它可以工作。

不过，在我看来，有一些限制。 我已经提到了运行aa和重命名局部变量所花费的时间，另外：

- Radare2不涉及面向对象的结构。 这意味着它不知道在DEX中实现了哪些类，或者哪些方法。 这使得难以解析样本。
  这就是为什么你阅读反汇编代码，直到你发现有趣的东西与r2不兼容。 相反，您将搜索特定的内容（URL，密码，对给定函数的调用）并深入研究。
- 入口点检测（命令ie）对Dalvik不起作用。 问题是Radare2不知道清单，所以它怎么能真正找到主要的活动呢？
  它确实发现了入口点，但只有低级别的入口点，如对android.support.v4.app.Fragment的onCreate（）的调用。

此外，在使用Radare2时，我建议使用大屏幕：每行代码一般都很长。甚至有种视觉模式，每个文本博客都以图形方式显示，但我并不热衷于此。
（我不明白为什么当JD，JEB2和IDA Pro可供喜欢GUI的人使用时，为什么人们会使用Radare2进行GUI - 但这是个人观点。）

总而言之，我会说我喜欢Radare2，因为它基于命令行，接近代码，可编写脚本。 但我发现很难对样本进行概述并解析它。
对于任何Radare2粉，我肯定会建议使用Radare2 for Android示例。 它运作良好。 但是，如果您是新手，门槛可能太高了。
如果您正在为Android寻找壳，我宁愿推荐Androguard。 如果您喜欢GUI，Radare2肯定不是选项，您应该坚持使用JD或JEB。

7. 结论/要点

以下是本文的内容：

- 要共享您的Android逆向工程框架，请考虑使用Docker镜像。 我的镜像可以通过`docker pull cryptax/android-re`获得，并且可以从[14]下载适应它的源。
- 要为您的Android示例编写字符串反混淆器，您可以从我的Ztorg去混淆器开始您自己的代码，可在[23]处获得。
- Android程序调试尚未运行。 CodeInspect和JEB2都很有希望，并且有望在未来几个月内实现，但在撰写本文时它们并不能很好地运作。
- Mitmproxy可用于窃听Android程序的加密通信。 设置需要在手机上添加新的CA证书并指定要使用的代理。
- Radare2可用于反转Dalvik可执行文件。 它还可以用于高级分析，例如字符串混淆（代码见[33]）。
  但是，学习如何使用Radare2在开始时有点困难，所以除非有一个非常具体的理由使用Radare2，否则新手可能会更好地坚持使用通常的apktool / baksmali / Java Decompiler（JD）。

参考

[1] Oliva Fora, P. Beginners Guide to Reverse Engineering Android Apps. In RSA Conference, February 2014. https://www.rsaconference.com/writable/presentations/file_upload/stu-w02b-beginners-guide-to-reverseengineering-android-apps.pdf.

[2] Altomare, D. Android Reverse Engineering 101. Parts 1 to 5. November 2015. http://www.fasteque.com/androidreverse-engineering-101-part-1/.

[3] Desnos, A.; Gueguen, G. Android: From Reversing to Decompilation. In BlackHat Abu Dhabi, 2011. https://media.blackhat.com/bh-ad-11/Desnos/bh-ad-11-DesnosGueguen-Andriod-Reversing_to_Decompilation_WP.pdf.

[4] Margaritelli, S. Android Applications Reversing 101. April 2017. https://www.evilsocket.net/2017/04/27/Android-Applications-Reversing-101/.

[5] Strazzere, T.; Sawyer, J. Android hacker protection level 0. In DEFCON 22, August 2014.

[6] Apvrille, A.; Nigam, R. Obfuscation in Android malware and how to fight back. In 8th International CARO Workshop, May 2014.

[7] Lipovsky, R. Obfuzzcation issues. In 8th International CARO Workshop, May 2014.

[8] Yu, R. Android packers: facing the challenges, building solutions. In Virus Bulletin International Conference, 2014. https://www.virusbulletin.com/virusbulletin/2016/01/paper-android-packers-facing-challenges-building-solutions/.

[9] http://redmine.honeynet.org/projects/are/wiki.

[10] https://github.com/sh4hin/Androl4b.

[11] https://www.docker.com/.

[12] Wikipedia. https://en.wikipedia.org/wiki/Docker_(software).

[13] Coleman, M. Containers are not VMs. March 2016. https://blog.docker.com/2016/03/containers-are-not-vms/.

[14] Dockerfile. https://github.com/cryptax/androidre.

[15] Rehm, F. Running GUI apps with Docker. September 2014. http://fabiorehm.com/blog/2014/09/11/running-gui-apps-withdocker/.

[16] Krijger, Q. Using supervisor with Docker to manage processes (supporting image inheritance). March 2014. http://blog.trifork.com/2014/03/11/using-supervisor-with-docker-tomanage-processes-supporting-imageinheritance/.

[17] Best practices for writing Dockerfiles. https://docs.docker.com/engine/userguide/eng-image/dockerfile_bestpractices/.

[18] PNF Software. Writing client scripts. https://www.pnfsoftware.com/jeb2/manual/dev/writing-client-scripts/.

[19] Falliere, N. Writing JEB2 scripts in Python. November 2015. https://www.pnfsoftware.com/blog/writingjeb2-scripts-in-python/.

[20] PNF Software. JEB API documentation. https://www.pnfsoftware.com/jeb2/apidoc/reference/packages.html.

[21] Apvrille, A. Teardown of a recent variant of Android/Ztorg – Part 1 and 2. March 2017. http://blog.fortinet.com/2017/03/15/teardown-of-a-recentvariant-of-android-ztorg-part-1 and http://blog.fortinet.com/2017/03/15/teardown-of-android-ztorg-part-2.

[22] https://github.com/pnfsoftware/jeb2-samplecode/tree/master/scripts.

[23] https://github.com/cryptax/misccode/blob/master/DeobfuscateZtorg.py.

[24] https://codeinspect.sit.fraunhofer.de/.

[25] http://docs.mitmproxy.org/en/stable/howmitmproxy.html.

[26] Wikipedia. https://en.wikipedia.org/wiki/Radare2.

[27] https://github.com/radare/radare2/issues/7376.

[28] Techorganic. Radare2 in 0x1e minutes. March 2016. https://blog.techorganic.com/2016/03/08/radare-2-in-0x1e-minutes/.

[29] A journey into Radare2 part 1. March 2017 https://www.megabeets.net/a-journeyinto-radare-2-part-1/.

[30] http://radare.tv.

[31] http://www.radare.org/r/talks.html.

[32] Pancake. Scripting r2 with pipes, May 2015. https://medium.com/@trufae/scripting-r2-with-pipes-47a7e14c50aa.

[33] https://github.com/cryptax/misccode/blob/master/r2ztorg.py.

上一篇：从Chrome源码看JavaScr... 下一篇：渗透测试工具备忘录

1. 0 条回复
   - 动动手指，沙发就是你的了！