|  |
| --- |
| 1、所谓黑色保活，就是利用不同的app进程**使用广播来进行相互唤醒**。  比如，利用系统产生的广播唤醒app（已经被Google取消）;接入第三方SDK也会唤醒相应的app进程（还有用）。  2、白色保活手段非常简单，就是调用系统api**启动一个前台的Service进程**，这样会在系统的**通知栏生成一个Notification**，用来让用户知道有这样一个app在运行着，哪怕当前的app退到了后台。常见比如，一些音乐播放器。  3、灰色保活，这种保活手段是应用范围最广泛。它是利用系统的漏洞来启动一个前台的Service进程，与普通的启动方式区别在于，它不会在系统通知栏处出现一个Notification，看起来就如同运行着一个后台Service进程一样。这样做带来的好处就是，用户无法察觉到你运行着一个前台进程（因为看不到Notification）,但你的进程优先级又是高于普通后台进程的。那么如何利用系统的漏洞呢，大致的实现思路和代码如下：  思路一：API < 18，启动前台Service时直接传入new Notification()；  思路二：API >=18，同时启动两个id相同的前台Service，然后再将后启动的Service做stop处理。  使用灰色保活并不代表着你的Service就永生不死了，只能说是提高了进程的优先级。  4、什么是oom\_adj？它是linux内核分配给每个系统进程的一个值，代表进程的优先级，进程回收机制就是根据这个优先级来决定是否进行回收。  进程的oom\_adj越大，表示此进程优先级越低，越容易被杀回收；越小，表示进程优先级越高，越不容易被杀回收。**普通app进程的oom\_adj>=0,系统进程的oom\_adj才可能<0。**  根据进程的重要性，可以划分为5级：  前台进程 (Foreground process)  可见进程 (Visible process)  服务进程 (Service process)  后台进程 (Background process)  空进程 (Empty process)  —————————————————————————————————  华为的机子上发现微信和手Q的UI进程退到后台，oom\_adj的值一点都没有变，因为微信在华为机子的白名单中。 |

当前业界的Android进程保活手段主要分为\*\* 黑、白、灰 \*\*三种，其大致的实现思路如下：

（1）黑色保活：不同的app进程，用广播相互唤醒（包括利用系统提供的广播进行唤醒）；

（2）白色保活：启动前台Service；

（3）灰色保活：利用系统的漏洞启动前台Service。

# 1.黑色保活

所谓黑色保活，就是利用不同的app进程使用广播来进行相互唤醒。举个3个比较常见的场景：

场景1：开机，网络切换、拍照、拍视频时候，利用系统产生的广播唤醒app。

场景2：接入第三方SDK也会唤醒相应的app进程，如微信sdk会唤醒微信，支付宝sdk会唤醒支付宝。由此发散开去，就会直接触发了下面的 场景3。

场景3：假如你手机里装了支付宝、淘宝、天猫、UC等阿里系的app，那么你打开任意一个阿里系的app后，有可能就顺便把其他阿里系的app给唤醒了。（只是拿阿里打个比方，其实BAT系都差不多）

没错，我们的Android手机就是一步一步的被上面这些场景给拖卡机的。

针对场景1，估计Google已经开始意识到这些问题，所以在最新的Android N取消了 ACTION\_NEW\_PICTURE（拍照），ACTION\_NEW\_VIDEO（拍视频），CONNECTIVITY\_ACTION（网络切换）等三种广播，无疑给了很多app沉重的打击。

而开机广播的话，记得有一些定制ROM的厂商早已经将其去掉。

针对场景2和场景3，因为调用SDK唤醒app进程属于正常行为，此处不讨论。但是在借助LBE分析app之间的唤醒路径的时候，发现了两个问题：

很多推送SDK也存在唤醒app的功能

app之间的唤醒路径真是多，且错综复杂

我把自己使用的手机测试结果给大家围观一下（我的手机是小米4C，刷了原生的Android5.1系统，且已经获得Root权限才能查看这些唤醒路径）。

可以看到以上3条唤醒路径，但是涵盖的唤醒应用总数却达到了23+43+28款，数目真心惊人。请注意，这只是我手机上一款app的唤醒路径而已，到了这里是不是有点细思极恐。

当然，这里依然存在一个疑问，就是LBE分析这些唤醒路径和互相唤醒的应用是基于什么思路，我们不得而知。所以我们也无法确定其分析结果是否准确，如果有LBE的童鞋看到此文章，不知可否告知一下思路呢？但是，手机打开一个app就唤醒一大批，我自己可是亲身体验到这种酸爽的......

# 2.白色保活

白色保活手段非常简单，就是调用系统api启动一个前台的Service进程，这样会在系统的通知栏生成一个Notification，用来让用户知道有这样一个app在运行着，哪怕当前的app退到了后台。如下方的LBE和QQ音乐这样。

# 3.灰色保活

灰色保活，这种保活手段是应用范围最广泛。它是利用系统的漏洞来启动一个前台的Service进程，与普通的启动方式区别在于，它不会在系统通知栏处出现一个Notification，看起来就如同运行着一个后台Service进程一样。这样做带来的好处就是，用户无法察觉到你运行着一个前台进程（因为看不到Notification）,但你的进程优先级又是高于普通后台进程的。那么如何利用系统的漏洞呢，大致的实现思路和代码如下：

思路一：API < 18，启动前台Service时直接传入new Notification()；

思路二：API >= 18，同时启动两个id相同的前台Service，然后再将后启动的Service做stop处理；

public class GrayService extends Service {

private final static int GRAY\_SERVICE\_ID = 1001;

@Override

public int onStartCommand(Intent intent, int flags, int startId) {

if (Build.VERSION.SDK\_INT < 18) {

startForeground(GRAY\_SERVICE\_ID, new Notification());//API < 18 ，此方法能有效隐藏Notification上的图标

} else {

Intent innerIntent = new Intent(this, GrayInnerService.class);

startService(innerIntent);

startForeground(GRAY\_SERVICE\_ID, new Notification());

}

return super.onStartCommand(intent, flags, startId);

}

...

...

/\*\*

\* 给 API >= 18 的平台上用的灰色保活手段

\*/

public static class GrayInnerService extends Service {

@Override

public int onStartCommand(Intent intent, int flags, int startId) {

startForeground(GRAY\_SERVICE\_ID, new Notification());

stopForeground(true);

stopSelf();

return super.onStartCommand(intent, flags, startId);

}

}}

代码大致就是这样，能让你神不知鬼不觉的启动着一个前台Service。其实市面上很多app都用着这种灰色保活的手段，什么？你不信？好吧，我们来验证一下。流程很简单，打开一个app，看下系统通知栏有没有一个 Notification，如果没有，我们就进入手机的adb shell模式，然后输入下面的shell命令

dumpsys activity services PackageName

打印出指定包名的所有进程中的Service信息，看下有没有 isForeground=true 的关键信息。如果通知栏没有看到属于app的 Notification 且又看到 isForeground=true 则说明了，此app利用了这种灰色保活的手段。

需要注意的是，使用灰色保活并不代表着你的Service就永生不死了，只能说是提高了进程的优先级。如果你的app进程占用了大量的内存，按照回收进程的策略，同样会干掉你的app。

# 4.进程回收机制

到这里基本就介绍完了\*\* 黑、白、灰 \*\*三种实现方式，仅仅从代码层面去讲保活是不够的，我希望能够通过系统的进程回收机制来理解保活，这样能够让我们更好的避免踩到进程被杀的坑。

熟悉Android系统的童鞋都知道，系统出于体验和性能上的考虑，app在退到后台时系统并不会真正的kill掉这个进程，而是将其缓存起来。打开的应用越多，后台缓存的进程也越多。在系统内存不足的情况下，系统开始依据自身的一套进程回收机制来判断要kill掉哪些进程，以腾出内存来供给需要的app。这套杀进程回收内存的机制就叫 Low Memory Killer ，它是基于Linux内核的 OOM Killer（Out-Of-Memory killer）机制诞生。

了解完 Low Memory Killer，再科普一下oom\_adj。什么是oom\_adj？它是linux内核分配给每个系统进程的一个值，代表进程的优先级，进程回收机制就是根据这个优先级来决定是否进行回收。对于oom\_adj的作用，你只需要记住以下几点即可：

进程的oom\_adj越大，表示此进程优先级越低，越容易被杀回收；越小，表示进程优先级越高，越不容易被杀回收。

根据进程的重要性，可以划分为5级：

前台进程 (Foreground process)

可见进程 (Visible process)

服务进程 (Service process)

后台进程 (Background process)

空进程 (Empty process)

————————————————

普通app进程的oom\_adj>=0,系统进程的oom\_adj才可能<0

那么我们如何查看进程的oom\_adj值呢，需要用到下面的两个shell命令

ps | grep PackageName //获取你指定的进程信息

这里是以我写的demo代码为例子，红色圈中部分别为下面三个进程的ID

UI进程：com.clock.daemon  
普通后台进程：com.clock.daemon:bg  
灰色保活进程：com.clock.daemon:gray

当然，这些进程的id也可以通过AndroidStudio获得

接着我们来再来获取三个进程的oom\_adj

cat /proc/进程ID/oom\_adj

从上图可以看到UI进程和灰色保活Service进程的oom\_adj=0，而普通后台进程oom\_adj=15。到这里估计你也能明白，为什么普通的后台进程容易被回收，而前台进程则不容易被回收了吧。但明白这个还不够，接着看下图

把app切换到后台，再进行一次oom\_adj的检验，你会发现UI进程的值从0变成了6,而灰色保活的Service进程则从0变成了1。这里可以观察到，app退到后台时，其所有的进程优先级都会降低。但是UI进程是降低最为明显的，因为它占用的内存资源最多，系统内存不足的时候肯定优先杀这些占用内存高的进程来腾出资源。所以，为了尽量避免后台UI进程被杀，需要尽可能的释放一些不用的资源，尤其是图片、音视频之类的。

有童鞋问，在华为的机子上发现微信和手Q的UI进程退到后台，oom\_adj的值一点都没有变，是不是有什么黑科技在其中。为此，我稍稍验证了一下，验证方式就是把demo工程的包名改成手机QQ的，编译运行在华为的机子上，发现我的进程怎么杀也都是不死的，退到后台oom\_adj的值同样不发生变化，而恢复原来的包名就不行了。所以，你懂的，手Q就在华为机子的白名单中。