# 1 反调试

[ANDROID调试检测技术汇编\_抚琴-CSDN博客](https://blog.csdn.net/qq_42186263/article/details/113711359)

[Anti-debugging Skills in APK - 国产大熊猫个人空间 - OSCHINA - 中文开源技术交流社区](https://my.oschina.net/9199771/blog/1584388)



## Ptrace检测

原理： 每个进程同时刻只能被1个进程ptrace调试

做法： 1. 主动ptrace自己，根据返回值判断自己是否被调试

2. 多进程ptrace

|  |
| --- |
| *// 单线程ptrace*  void ptraceCheck()  {  *// ptrace如果被调试返回值为-1，如果正常运行，返回值为0*      int iRet = ptrace(PTRACE\_TRACEME, 0, 0, 0);      if (-1 == iRet)      {          LOGA("ptrace失败，进程正在被调试\n");          return;      }      else      {          LOGB("ptrace的返回值为:%d\n", iRet);          return;      }  } |

Anti：frida replace ptrace函数

|  |
| --- |
| export function anti\_ptrace() {          var ptrace = Module.findExportByName(null, "ptrace");          if (null != ptrace) {              ptrace = ptrace.or(1);              log.i('anti\_ptrace', "ptrace addr: " + ptrace);  *// Interceptor.attach(ptrace, {*  *//     onEnter: function (args) {*  *//         log.i('anti\_ptrace', 'entry');*  *//     }*  *// });*              Interceptor.replace(ptrace.or(1), new NativeCallback(function (p1: any, p2: any, p3: any, p4: any) {                  log.i('anti\_ptrace', 'entry');                  return 1;              }, 'long', ['int', "int", 'pointer', 'pointer']));          }      } |

## 1.2 进程信息检查

[玩转Android10源码开发定制(六)修改内核源码绕过反调试检测\_xiaomaNo01的专栏-CSDN博客](https://blog.csdn.net/xiaomaNo01/article/details/111967193)

第一种：/proc/pid/status 和 /proc/pid/task/pid/status

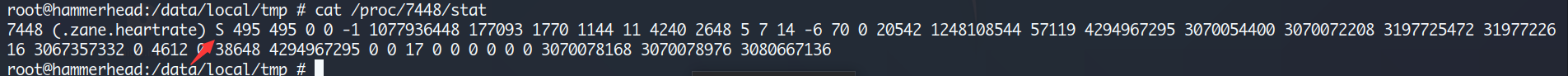
在Android 中调试状态下，linux内核会向/proc/$pid/status或者/proc/$pid/task/$pid/status 中写入进程状态信息。其中TracerPid 字段写入调试该进程的进程的的Pid。其中State字段中写入该进程当前处于的状态,取值如下之一:

R (running)", "S (sleeping)", "D (disk sleep)", "T (stopped)", "t(tracing stop)", "Z (zombie)",  "X (dead)"

当State为t (tracing stop)或者T (stopped)的时候,表示正被调试追踪。所以反调试的方法之一就是通过不断轮询读取/proc/$pid/status或者/proc/$pid/task/$pid/status文件检测检查TracerPid的值或者State的值，如果TracerPid非0说明该进程被调试;如果state值为t (tracing stop)或者T (stopped),说明被调试追踪。

第二种：/proc/pid/stat 和 /proc/pid/task/pid/stat

第二个字段是t（T）



第三种：/proc/pid/wchan 和 /proc/pid/task/pid/wchan

wchan文件内容表示显示当进程sleep时，kernel当前运行的函数。若进程被调试，内核会往/proc/$pid/wchan、/proc/$pid/task/$pid/wchan文件中写入ptrace\_stop信息。

|  |
| --- |
| */\*\**  *\* @state\_name: cat /proc/xxx/stat ==> ...(<state\_name>) S...*  *\**  *\* anti fgets function include :*  *\* status->TracerPid, SigBlk, S (sleeping)*  *\* State->(package) S*  *\* wchan->SyS\_epoll\_wait*  *\*/*      export function anti\_fgets() {          const tag = 'anti\_fgets';          const fgetsPtr = Module.findExportByName(null, 'fgets');          log.i(tag, 'fgets addr: ' + fgetsPtr);          if (null == fgetsPtr) {              return;          }          var fgets = new NativeFunction(fgetsPtr, 'pointer', ['pointer', 'int', 'pointer']);          Interceptor.replace(fgetsPtr, new NativeCallback(function (buffer, size, fp) {              if (null == this) {                  return 0;              }              var logTag = null;  *// 进入时先记录现场*              const lr = fridaCommon.getLR(this.context);  *// 读取原 buffer*              var retval = fgets(buffer, size, fp);              var bufstr = (buffer as NativePointer).readCString();              if (null != bufstr) {                  if (bufstr.indexOf("TracerPid:") > -1) {                      buffer.writeUtf8String("TracerPid:\t0");                      logTag = 'TracerPid';                  }  *//State:    S (sleeping)*                  else if (bufstr.indexOf("State:\tt (tracing stop)") > -1) {                      buffer.writeUtf8String("State:\tS (sleeping)");                      logTag = 'State';                  }  *// ptrace\_stop*                  else if (bufstr.indexOf("ptrace\_stop") > -1) {                      buffer.writeUtf8String("sys\_epoll\_wait");                      logTag = 'ptrace\_stop';                  }  *//(sankuai.meituan) t*                  else if (bufstr.indexOf(") t") > -1) {                      buffer.writeUtf8String(bufstr.replace(") t", ") S"));                      logTag = 'stat\_t';                  }  *// SigBlk*                  else if (bufstr.indexOf('SigBlk:') > -1) {                      buffer.writeUtf8String('SigBlk:\t0000000000001204');                      logTag = 'SigBlk';                  }                  if (logTag) {                      log.i(tag + " " + logTag, bufstr + " -> " + buffer.readCString() + ' lr: ' + lr                          + "(" + fridaCommon.getModuleByAddr(lr) + ")");                  }              }              return retval;          }, 'pointer', ['pointer', 'int', 'pointer']));      } |

## IDA调试端口检测

原理：调试器远程调试时，会占用一些固定的端口号。

做法：读取/proc/net/tcp，查找IDA远程调试所用的23946端口，若发现说明进程正在被IDA调试。（也可以运行netstat ­apn结果中搜索23946端口）

|  |
| --- |
| void CheckPort23946ByTcp()  {      FILE \*pfile = NULL;      char buf[0x1000] = {0};  *// 执行命令*      char \*strCatTcp = "cat /proc/net/tcp |grep :5D8A";  *//char\* strNetstat="netstat |grep :23946";*      pfile = popen(strCatTcp, "r");      if (NULL == pfile)      {          LOGA("CheckPort23946ByTcp popen打开命令失败!\n");          return;      }  *// 获取结果*      while (fgets(buf, sizeof(buf), pfile))      {  *// 执行到这里，判定为调试状态*          LOGA("执行cat /proc/net/tcp |grep :5D8A的结果:\n");          LOGB("%s", buf);      } *//while*      pclose(pfile);  } |

## 调试器进程名检测

原理：远程调试要在手机中运行android\_server gdbserver gdb等进程。

做法：遍历进程，查找固定的进程名，找到说明调试器在运行。

|  |
| --- |
| void SearchObjProcess()  {      FILE \*pfile = NULL;      char buf[0x1000] = {0};  *// 执行命令*  *//pfile=popen("ps | awk '{print $9}'","r"); // 部分不支持awk命令*      pfile = popen("ps", "r");      if (NULL == pfile)      {          LOGA("SearchObjProcess popen打开命令失败!\n");          return;      }  *// 获取结果*      LOGA("popen方案:\n");      while (fgets(buf, sizeof(buf), pfile))      {  *// 打印进程*          LOGB("遍历进程:%s\n", buf);  *// 查找子串*          char \*strA = NULL, strB = NULL, strC = NULL, strD = NULL;          strA = strstr(buf, "android\_server");          strB = strstr(buf, "gdbserver");          strC = strstr(buf, "gdb");          strD = strstr(buf, "fuwu");          if (strA || strB || strC || strD)          {  *// 执行到这里，判定为调试状态*              LOGB("发现目标进程:%s\n", buf);          } *//if*      }     *//while*      pclose(pfile);  } |

# 2 JEB脚本开发

[三分钟上手JEB脚本开发 - 安全客，安全资讯平台 (anquanke.com)](https://www.anquanke.com/post/id/228981#h3-6)

[IJavaMethod | JEB API Documentation (pnfsoftware.com)](https://www.pnfsoftware.com/jeb/apidoc/reference/com/pnfsoftware/jeb/core/units/code/java/IJavaMethod.html)

[jeb-samplecode/scripts at master · pnfsoftware/jeb-samplecode · GitHub](https://github.com/pnfsoftware/jeb-samplecode/tree/master/scripts)

[[原创]JEB脚本(一)(指令解析 反编译 抽象语法树)-Android安全-看雪论坛-安全社区|安全招聘|bbs.pediy.com](https://bbs.pediy.com/thread-263011.htm#msg_header_h1_4)

# 3 IDA

## 3.1 结构体

[7.IDA-创建结构体\_liujiayu2的专栏-CSDN博客\_ida 结构体](https://blog.csdn.net/liujiayu2/article/details/77488078)

[[IDA] 在IDA里如何自定义数据结构\_盗世九昌的技术博客\_51CTO博客](https://blog.51cto.com/u_13992485/2476125)

# 4 情报研究网站

[研究报告-奇安信 (qianxin.com)](https://www.qianxin.com/threat/reportaptlist)

[360研究报告(安全研究列表页)，从数字中感知威胁，在对抗中排除隐患。基于360安全大数据，进行分析研究，将安全可视化，让大众了解数字背后的含义](http://zt.360.cn/2015/reportlist.html?list=1)

[信息安全知识库 vipread.com](https://vipread.com/index)