多种特征检测 Frida

转载

双刃剑客 于 2020-11-14 21:58:13 发布 2029 收藏 6
分类专栏： Frida
原文链接：https://bbs.pediy.com/thread-263438.htm

引用：https://bbs.pediy.com/thread-263438.htm

之前看雪上读到过一篇检测 frida 的文章(参考链接在文末)。这里我们做了如下两步来反检测:

1、更名: 将frida-server改名为fenfeiserver

2、改端口: 手机里面用 fenfeiserver -l 127.0.0.1:8080 来启动; 电脑里面先映射 adb forward tcp:8080 tcp:8080; 然后启动 frida -H 127.0.0.1:8080 -l jd.js com.XXXXX.mall

转：https://bbs.pediy.com/thread-217482.htm

多种特征检测 Frida

Frida 在逆向工程狮中很受欢迎，你基本可以在运行时访问到你能想到的任何东西，内存地址、native 函数、Java 实例对象等。在 OWASP 的移动测试指南里就提到了 Frida。但是啊，每出来个好用的注入工具，都会有反注入、反反注入、反反反注入、反...注入。这篇文章要介绍的是 Android APP 检测 Frida 的方法。

检查 Frida 的痕迹

一种简易方法是检测 Frida 的运行痕迹，也适用于同类工具的检测，比如包文件、二进制文件、库文件、进程、临时文件等等。本例中针对的对象是 fridaserver，它通过 TCP 对外与 frida 通信，此时可以用 Java 遍历运行的进程列表从而检查 fridaserver 是否在运行。

```
public boolean checkRunningProcesses() {
  boolean returnValue = false;
  // Get currently running application processes
  List<RunningServiceInfo> list = manager.getRunningServices(300);
  if(list != null){
    String tempName;
    for(int i=0;i<list.size();++i){
      tempName = list.get(i).process;
      if(tempName.contains("fridaserver")) {
        returnValue = true;
      }
    }
  }
  return returnValue;
}
```

若 Frida 运行在默认配置时此法有效，若是遇到个笨拙的脚本小子，在第一步就能绊倒他。绕过也是相当简单，只需重命名 fridaserver，我们得找个更好的方法。

fridaserver 默认的 TCP 端口是 27047，可以检查这个端口是否开放。native 代码如下：

```
boolean is_frida_server_listening() {
    struct sockaddr_in sa;
    memset(&sa, 0, sizeof(sa));
    sa.sin_family = AF_INET;
    sa.sin_port = htons(27047);
    inet_aton("127.0.0.1", &(sa.sin_addr));
    int sock = socket(AF_INET , SOCK_STREAM , 0);
    if (connect(sock , (struct sockaddr*)&sa , sizeof sa) != -1) {
      /* Frida server detected. Do something… */
    }
}
```

同样，这也得要求 fridaserver 是默认配置运行，命令行指定参数就可以改变它的监听端口，绕过也太不麻烦了。不过我们可以用' nmap -sV' 找到开放端口来改善这个方法。因为 fridaserver 使用 D-Bus 协议通信，我们为每个开放的端口发送 D-Bus 的认证消息，哪个端口回复了哪个就是 fridaserver。

```
/*
 * Mini-portscan to detect frida-server on any local port.
 */
for(i = 0 ; i <= 65535 ; i++) {
    sock = socket(AF_INET , SOCK_STREAM , 0);
    sa.sin_port = htons(i);
    if (connect(sock , (struct sockaddr*)&sa , sizeof sa) != -1) {
        __android_log_print(ANDROID_LOG_VERBOSE, APPNAME,  "FRIDA DETECTION [1]: Open Port: %d", i);
        memset(res, 0 , 7);
        // send a D-Bus AUTH message. Expected answer is “REJECT"
        send(sock, "\x00", 1, NULL);
        send(sock, "AUTH\r\n", 6, NULL);
        usleep(100);
        if (ret = recv(sock, res, 6, MSG_DONTWAIT) != -1) {
            if (strcmp(res, "REJECT") == 0) {
               /* Frida server detected. Do something… */
            }
        }
    }
    close(sock);
}
```

我们现在好像有了个非常好用的方法了呢，但是还存在问题。Frida 提供不需要 fridaserver 运行的模式！怎么检测？！

Frida 的各个模式都是用来注入的，我们可以利用的点就是 frida 运行时映射到内存的库。最直接的是挨个检查加载的库。

```
char line[512];
FILE* fp;
fp = fopen("/proc/self/maps", "r");
if (fp) {
    while (fgets(line, 512, fp)) {
        if (strstr(line, "frida")) {
            /* Evil library is loaded. Do something… */
        }
    }
    fclose(fp);
    } else {
       /* Error opening /proc/self/maps. If this happens, something is off. */
    }
}
```

这段代码检测名字含有“frida”的库，表明上有用，实际上：

- 还记得为什么检测名字是“fridaserver”的方法为什么不可靠吧？这里也是一样，稍微改一下 frida 就能重命名代理库名。

- 这段代码依赖的是标准库的`fopen()`和`strstr()`函数，可笑的是，我们竟想用能被 frida 轻而易举就 hook 的函数来检测 frida！

问题1可以用经典的病毒扫描法解决，在内存中扫描 frida 的库特征 “gadgets”。我选择字符串 “LIBFRIDA”，它在所有 frida-gadget 和 frida-agent 的版本中都有出现。下面的代码扫描了在 `/proc/sel/maps` 里找到的所有的可执行段，为了简洁我放了部分代码，完整的在 https://github.com/b-mueller/frida-detection-demo/blob/master/AntiFrida/app/src/main/cpp/native-lib.cpp 。

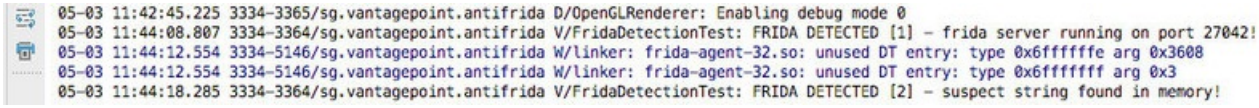
```
static char keyword[] = "LIBFRIDA";
num_found = 0;
int scan_executable_segments(char * map) {
    char buf[512];
    unsigned long start, end;
    sscanf(map, "%lx-%lx %s", &start, &end, buf);
    if (buf[2] == 'x') {
        return (find_mem_string(start, end, (char*)keyword, 8) == 1);
    } else {
        return 0;
    }
}
void scan() {
    if ((fd = my_openat(AT_FDCWD, "/proc/self/maps", O_RDONLY, 0)) >= 0) {
    while ((read_one_line(fd, map, MAX_LINE)) > 0) {
        if (scan_executable_segments(map) == 1) {
            num_found++;
        }
    }
    if (num_found > 1) {
        /* Frida Detected */
    }
}
```

注意 `my_openat()` 等函数，它们并非平常的 libc 库函数，是自定义实现的，但是功能和 libc 中的一样，设置了系统调用的参数，执行了软中断。因为直接调用公共 API 并不可靠，这样不容易被 hook。完整的实现在 https://github.com/b-mueller/frida-detection-demo/blob/master/AntiFrida/app/src/main/cpp/syscall.S 。下面是 my_openat 的代码：

```
#include "bionic_asm.h"
.text
    .globl my_openat
    .type my_openat,function
my_openat:
    .cfi_startproc
    mov ip, r7
    .cfi_register r7, ip
    ldr r7, =__NR_openat
    swi #0
    mov r7, ip
    .cfi_restore r7
    cmn r0, #(4095 + 1)
    bxls lr
    neg r0, r0
    b __set_errno_internal
    .cfi_endproc
    .size my_openat, .-my_openat;
```

到这里总算是有效的方法了，只用 frida 的话也不容易绕过，加了混淆更难。即使这样，依然有很多办法可以绕过，直接能想到的就是打补丁、hook 系统调用。但是记住，**逆向工程永远胜利！**

想要试验，可以在这里 https://github.com/b-mueller/frida-detection-demo/ 下载 Android studio 工程。frida 注入时的运行结果如下：



原文链接：http://www.vantagepoint.sg/blog/90-the-jiu-jitsu-of-detecting-frida

本文由 看雪翻译小组 kiyaa 编译