# Xhook

xhook 是一个针对 Android 平台 ELF (可执行文件和动态库) 的 PLT (Procedure Linkage Table) hook 库。

xhook 一直在稳定性和兼容性方面做着持续的优化。

## 特征

* 支持 Android 4.0 (含) 以上版本 (API level >= 14)。
* 支持 armeabi，armeabi-v7a，arm64-v8a，x86 和 x86\_64。
* 支持 **ELF HASH** 和 **GNU HASH** 索引的符号。
* 支持 **SLEB128** 编码的重定位信息。
* 支持通过正则表达式批量设置 hook 信息。
* **不**需要 ROOT 权限。
* 不依赖于任何的第三方动态库。
* 纯 C 的代码。比较小的库体积

## 编译

需要 google NDK 来编译 xhook。  
<https://developer.android.com/ndk/downloads/index.html>

最新版本的 xhook 在开发和调试中使用的 NDK 版本是 **r16b**

* 编译动态库 (libxhook.so 和其他的用于测试的动态库)

./build\_libs.sh

* 把动态库安装到 Demo 工程的 libs 目录中

./install\_libs.sh

* 清除动态库

./clean\_libs.sh

## Demo

D

cd ./xhookwrapper/

./gradlew assembleDebug

adb install ./app/build/outputs/apk/debug/app-debug.apk

## API

外部 API 头文件: libxhook/jni/xhook.h

### 1. 注册 hook 信息

F

int xhook\_register(const char \*pathname\_regex\_str,

const char \*symbol,

void \*new\_func,

void \*\*old\_func);

在当前进程的内存空间中，在每一个符合正则表达式 pathname\_regex\_str 的已加载ELF中，每一个调用 symbol 的 PLT 入口点的地址值都将给替换成 new\_func。之前的 PLT 入口点的地址值将被保存在 old\_func 中。

new\_func 必须具有和原函数同样的函数声明。

成功返回 0，失败返回 非0。

pathname\_regex\_str 只支持 **POSIX BRE (Basic Regular Expression)** 定义的正则表达式语法。

### . 忽略部分 hook 信息

int xhook\_ignore(const char \*pathname\_regex\_str,

const char \*symbol);

根据 pathname\_regex\_str 和 symbol，从已经通过 xhook\_register 注册的 hook 信息中，忽略一部分 hook 信息。如果 symbol 为 NULL，xhook 将忽略所有路径名符合正则表达式 pathname\_regex\_str 的 ELF。

成功返回 0，失败返回 非0。

pathname\_regex\_str 只支持 **POSIX BRE** 定义的正则表达式语法。

### 3. 执行 hook

int xhook\_refresh(int async);

根据前面注册的 hook 信息，执行真正的 hook 操作。

给 async 参数传 1 表示执行异步的 hook 操作，传 0 表示执行同步的 hook 操作。

成功返回 0，失败返回 非0。

xhook 在内部维护了一个全局的缓存，用于保存最后一次从 /proc/self/maps 读取到的 ELF 加载信息。每次一调用 xhook\_refresh 函数，这个缓存都将被更新。xhook 使用这个缓存来判断哪些 ELF 是这次新被加载到内存中的。我们每次只需要针对这些新加载的 ELF 做 hook 就可以了。

### 清除缓存

void xhook\_clear();

清除 xhook 的缓存，重置所有的全局标示。

如果你确定你需要的所有 PLT 入口点都已经被替换了，你可以调用这个函数来释放和节省一些内存空间。

### 5. 启用/禁用 调试信息

void xhook\_enable\_debug(int flag);

给 flag 参数传 1 表示启用调试信息，传 0 表示禁用调试信息。 (默认为：**禁用**)

调试信息将被输出到 logcat，对应的 TAG 为：xhook。

### 6. 启用/禁用 SFP (段错误保护)

void xhook\_enable\_sigsegv\_protection(int flag);

给 flag 参数传 1 表示启用 SFP，传 0 表示禁用 SFP。 (默认为：**启用**)

xhook 并不是一个常规的业务层的动态库。在 xhook 中，我们不得不直接计算一些内存指针的值。在一些极端的情况和环境下，读或者写这些指针指向的内存会发生段错误。根据我们的测试，xhook 的行为将导致 APP 崩溃率增加 “一千万分之一” (0.0000001)。（具体崩溃率可能会增加多少，也和你想要 hook 的库和符号有关）。最终，我们不得不使用某些方法来防止这些无害的崩溃。我们叫它SFP (段错误保护)，它是由这些调用和值组成的：sigaction()， SIGSEGV， siglongjmp() 和 sigsetjmp()。

**在 release 版本的 APP 中，你应该始终启用 SFP，这能防止你的 APP 因为 xhook 而崩溃。在 debug 版本的 APP 中，你应该始终禁用 SFP，这样你就不会丢失那些一般性的编码失误导致的段错误，这些段错误是应该被修复的。**

## 应用举例

Fg

//监测内存泄露

xhook\_register(".\*\\.so$", "malloc", my\_malloc, NULL);

xhook\_register(".\*\\.so$", "calloc", my\_calloc, NULL);

xhook\_register(".\*\\.so$", "realloc", my\_realloc, NULL);

xhook\_register(".\*\\.so$", "free", my\_free, NULL);

//监控 sockets 生命周期

xhook\_register(".\*\\.so$", "getaddrinfo", my\_getaddrinfo, NULL);

xhook\_register(".\*\\.so$", "socket", my\_socket, NULL);

xhook\_register(".\*\\.so$", "setsockopt" my\_setsockopt, NULL);

xhook\_register(".\*\\.so$", "bind", my\_bind, NULL);

xhook\_register(".\*\\.so$", "listen", my\_listen, NULL);

xhook\_register(".\*\\.so$", "connect", my\_connect, NULL);

xhook\_register(".\*\\.so$", "shutdown", my\_shutdown, NULL);

xhook\_register(".\*\\.so$", "close", my\_close, NULL);

//过滤出和保存部分安卓 log 到本地文件

xhook\_register(".\*\\.so$", "\_\_android\_log\_write", my\_log\_write, NULL);

xhook\_register(".\*\\.so$", "\_\_android\_log\_print", my\_log\_print, NULL);

xhook\_register(".\*\\.so$", "\_\_android\_log\_vprint", my\_log\_vprint, NULL);

xhook\_register(".\*\\.so$", "\_\_android\_log\_assert", my\_log\_assert, NULL);

//追踪某些调用 (忽略 linker 和 linker64)

xhook\_register("^/system/.\*$", "mmap", my\_mmap, NULL);

xhook\_register("^/vendor/.\*$", "munmap", my\_munmap, NULL);

xhook\_ignore (".\*/linker$", "mmap");

xhook\_ignore (".\*/linker$", "munmap");

xhook\_ignore (".\*/linker64$", "mmap");

xhook\_ignore (".\*/linker64$", "munmap");

//防御某些注入攻击

xhook\_register(".\*com\\.hacker.\*\\.so$", "malloc", my\_malloc\_always\_return\_NULL, NULL);

xhook\_register(".\*/libhacker\\.so$", "connect", my\_connect\_with\_recorder, NULL);

//修复某些系统 bug

xhook\_register(".\*some\_vendor.\*/libvictim\\.so$", "bad\_func", my\_nice\_func, NULL);

//忽略 libwebviewchromium.so 的所有 hook 信息

xhook\_ignore(".\*/libwebviewchromium.so$", NULL);

//现在执行 hook!

xhook\_refresh(1);

## 参考

https://github.com/iqiyi/xHook/blob/master/README.zh-CN.md