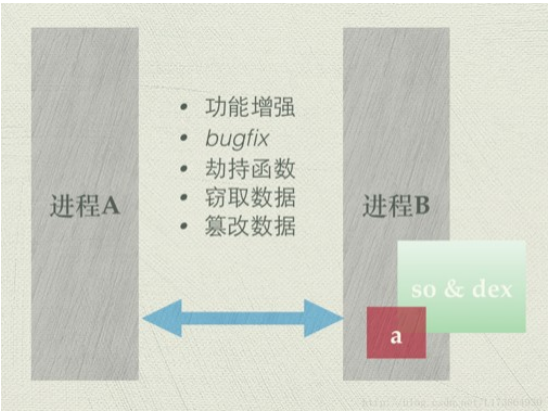
下一步：我们可以在 Python 和 C++侧加入对内存的监控，以解决跨语言内存泄漏难以调试的问题

动态代码注入 overview

动态代码注入，针对是进程，比如修改进程的寄存器、内存值等等；动态跟静态最大的区别是，动态不需要改动源文件，但需要高权限（通常是 root 权限）

动态注入技术，本质上就是一种调度技术。功能：查看变量值，修改变量值，跟踪进程跳转，查看进程调用堆栈等等。动态注入相比于普通的调试，最大的区别就是动态注入是一个**"自动化调试并达到加载自定义动态链接库"**的过程。所谓自动化，其实就是通过代码实现，在 Linux 上通过 Ptrace 就可以完成上面所有功能，当然 Ptrace 功能是比较原始的，平时调试中的功能还需要很多高层逻辑封装才可以实现

过程：
如下图所示，进程 A 注入到进程 B 后，通过修改寄存器和内存，让进程 B 加载自定义的动态库 a，当 a 被加载后，a 会尝试加载其他模块，比如加载 dex 文件等等，具体的注入过程如下：
ATTATCH，指定目标进程，开始调试；
GETREGS，获取目标进程的寄存器，保存现场；
SETREGS，修改 PC 等相关寄存器，使其指向 mmap；
POPETEXT，把 so path 写入 mmap 申请的地址空间；
SETRESG，修改 PC 等相关寄存器，使其指向 dlopen；
SETREGS，恢复现场；DETACH，解除调试，使其恢复；



代码：https://github.com/boyliang/Poison
感谢 https://blog.csdn.net/l173864930/article/details/38456313

在 Linux 上动态代码注入

利用 **linux-inject**：一个注入程序，可以注入一个.so 文件到一个运行中的应用程序进程中。类似于 LD_PRELOAD 环境变量所实现的功能，但它可以在程序运行过程中进行动态注入，而 LD_PRELOAD 是定义在程序运行前优先加载的动态链接库
实例教程：https://blog.csdn.net/hpp24/article/details/52125568

利用 **ptrace**:它提供了父进程可以观察和控制其子进程执行的能力，并允许父进程检查和替换子进程的内核镜像(包括寄存器)的值。其基本原理是：当使用了 ptrace 跟踪后，所有发送给被跟踪的子进程的信号(除了 SIGKILL)，都会被转发给父进程，而子进程则会被阻塞，这时子进程的状态就会被系统标注为 TASK_TRACED。而父进程收到信号后，就可以对停止。ptrace 可以实时监测和修改另一个进程的运行，它是如此的强大以至于曾经因为它在 unix-like 平台(如 Linux，*BSD)上产生了各种漏洞
实例教程：https://blog.csdn.net/Dearggae/article/details/47379245

利用**/proc 文件系统**：避免用 ptrace；/proc 文件系统提供了对 Linux 系统运行的**内省**(Introspection)，每个进程在文件系统中都有自己的目录，其中包含有关流程及其内部的详细信息。在这个目录中，有两个伪文件，分别是 maps 文件和 mem 文件。
maps 文件包含分配给二进制文件的所有内存区域架构，以及所有包含的动态库。不过，这个信息现在相对敏感，因为每个库位置的偏移量是由 ASLR 随机化的。
mem 文件提供了流程使用的完整内存空间的稀疏架构，结合从 maps 文件获得的偏移量，可以使用 mem 文件读取和写入进程的内存空间。如果偏移量是错误的，或者从开始位置按顺序读取文件，将返回读写错误，因为这相当于是读取不可访问的未分配内存
注：内省（Introspection）是面向对象语言和环境的一个强大特性，它是对象揭示自己作为一个运行时对象的详细信息的一种能力。这些详细信息包括对象在继承树上的位置，对象是否遵循特定的协议，以及是否可以响应特定的消息
详细介绍：https://yq.aliyun.com/articles/214065

方案：
本调研仅提供下一步对内存的监控工作中关于动态代码注入的现有工具和思路。由于缺乏和组员交流，对已完成的成功仅停留在整体框架，没有深入细节，故没法立马提出可行方案。希望后面能和大家讨论。

附录（**可忽略**）：
Android so 库注入代码：http://bbs.pediy.com/showthread.php?t=141355
代码解读：https://blog.csdn.net/qq1084283172/article/details/46859931

Androd so 注入和函数 Hook（基于 got 表）的步骤
1.ptrace 附加目标 pid 进程；
2.在目标 pid 进程中，查找内存空间（用于存放被注入的 so 文件的路径和 so 中被调用的函数的名称或者 shellcode）；
3.调用目标 pid 进程中的 dlopen、dlsym 等函数，用于加载 so 文件实现 Android so 的注入和函数的 Hook；

4.释放附加的目标 pid 进程和卸载注入的 so 文件

一、在目标 pid 进程中查找内存空间的方法

方法一：获取目标 pid 进程加载的"/system/lib/libc.so"动态库文件中 mmap 函数的调用地址，然后远程调用目标 pid 进程中的 mmap 函数，在目 pid 进程中申请内存空间用于保存被注入加载的 so 库文件的路径的名称和被加载的 so 库文件中被调用的导出函数，主要为后面调用 dlopen 加载 so 库文件和调用 dlsym 获取被注入的 so 文件的中导出函数做准备即为后面进行 Android 的 so 注入和函数的 Hook 做准备（见作者 ariesjzj 提供的 Android 的 so 注入和函数 Hook 的思路）或者向目标 pid 进程中写入 shellcode 操作。

方法二：遍历目标 pid 进程的/proc/pid/maps 文件，找到空闲的内存空间，用于存放调用 dlopen、dlsym 等函数的参数或者执行的 shellcode 代码

二、在目标 pid 进程中加载 so 实现 Android 的 so 注入和函数 Hook 的方法

方法一：初期 Android so 的注入版本（古河）的 Android 的 so 的注入和函数 Hook 是由 arm 汇编的 shellcode 实现，代码的移植性不好而且不易阅读，具体的实现就是：将 dlopen、dlsym 等函数调用需要的函数参数以及执行 so 加载和执行 Hook 目标 pid 进程中的函数的 shellcode 代码指令写入到目标 pid 进程内存中，然后修改目标 pid 进程的 PC 指令计数寄存器跳转到目标 pid 进程内存的 shellcode 处执行加载 so 和 Hook 目标 pid 进程的函数(基于 got 表）。

方法二：其实呢，注入 so 库文件到目标 pid 进程以及 Hook 目标 pid 进程中 got 表的函数的实现，不需要用注入 shellcode 到目标 pid 进程中的方式来解决。通过获取目标 pid 进程中 dlopen、dlsym 等函数的调用地址，远程调用目标 pid 进程的 dlpoen 函数可以实现 so 库文件的注入，远程调用目标 pid 进程中 dlsym 函数获取加载的 so 库文件中的导出函数，调用该导出函数对目标 pid 进程中基于 got 表的目标函数进行 Hook 即可。

三、在目标 pid 进程中注入 so 以后进行一次函数调用，实现 Hook 目标函数

方法一：前面的步骤中已经将 so 库文件注入到目标 pid 进程中了，只需要远程调用目标 pid 进程中的 dlsym 函数获取执行函数 Hook 的导出函数，然后远程调用目标 pid 进程中该导出函数即可实现 Hook 目标 pid 进程中基于 got 表的目标函数。

方法二：不需要调用目标 pid 进程中 dlsym 函数获取注入 so 文件的导出函数，来实现 Hook 目标 pid 进程的目标函数。在 so 库文件加载的时候，会首先执行.init 段的构造函数，该构造函数的定义方法为 void __attribute__((constructor)) x_init(void)，因此当我们向目标 pid 进程注入 so 库文件的时候，执行该 x_init 函数，可以实现 Hook 目标 pid 进程的函数的目的，该 x_init 函数唯一的不足就是不能传递函数参数。

方法三：在注入到目标 pid 进程中的 so 库文件的源码文件中定义全局变量，在 so 库文件加载到目标 pid 进程的内存中，全局变量的初始化的时候，有一次执行 Hook 目标 pid 进程的基于 got 表的目标函数的机会。也就是在注入的 so 的源码文件中定义一个全局变量(对象)，当全局变量初始化时（全局对象变量调用构造函数的时候），有一次执行 Hook 目标函数的机会即使用 c++全局对象初始化，其构造函数会被自动执行。

四、在目标 pid 进程中 Hook 目标函数的实现方法

方法一：通过解析目标 pid 进程内存中的需要被 Hook 的目标 so 库文件，找到该目标 so 文件的.got 表，然后遍历查找到需要被 Hook 的目标函数的调用地址进行替换，替换为我们自定义的函数的地址。

方法二：获取将被 Hook 的 so 库文件内存加载后产生的 soinfo 结构体指针，而该结构体保存着该目标 so 文件加载到内存中的各种信息,然后也是遍历目标 so 文件的 got 表找到将被 Hook 的目标函数的调用地址进行挂钩 Hook 替换为我自定义的函数。

感谢张昱老师找到的 https://blog.csdn.net/QQ1084283172/article/details/54095995