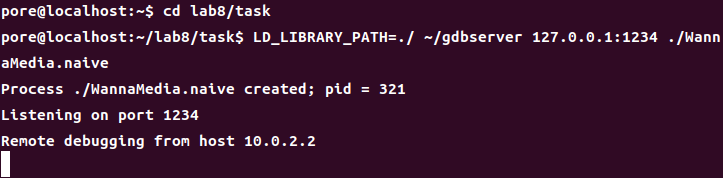
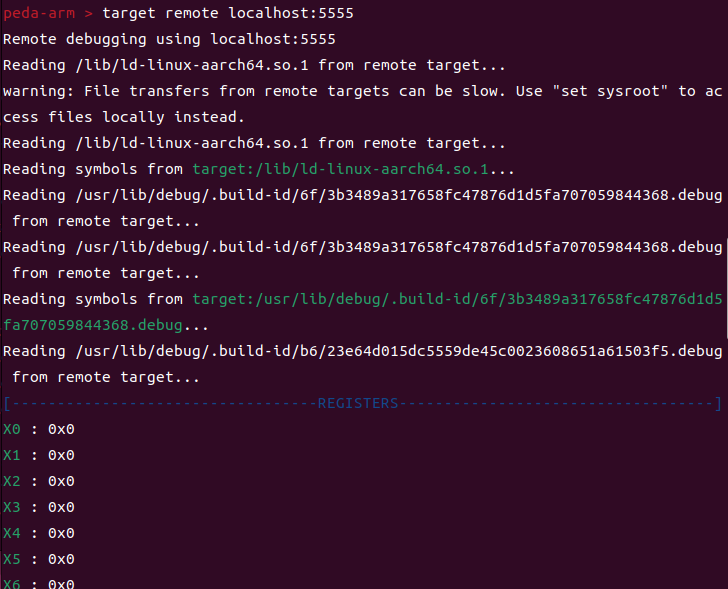
任务一

步骤一



启动gdbserver

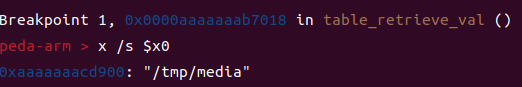


gdb-multiarch WannaMedia.naive

#等待调试器启动完成

peda-arm> target remote localhost:5555 #localhost:5555即QEMU中1234，详见QEMU虚拟机启动文件--start\_vm.sh

步骤三：



任务二

通过info function来定位0xaaaaaaab7018是哪个函数的位置，找到函数是

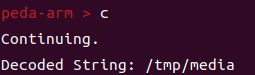
所以函数的开头地址是0xaaaaaaab6f9c，在末尾加上FuncStart(‘\*0xaaaaaaab6f9c’)



最后打印出来的开头参数：



结尾参数：

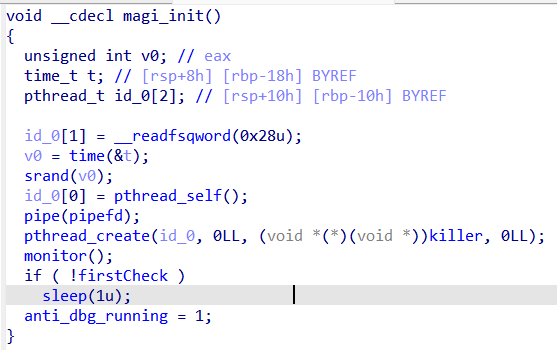


gdb.execute("run")不需要添加，原因有二：

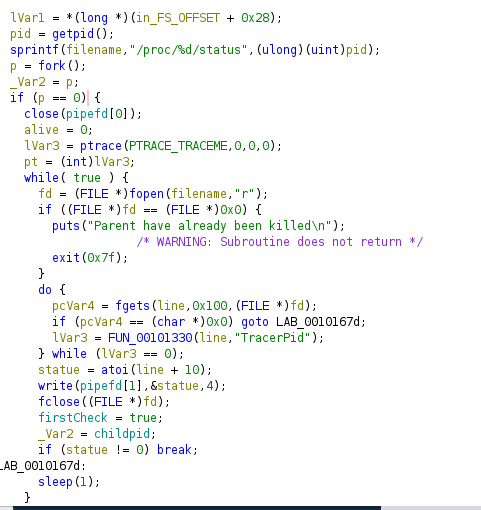
1.target remote不支持run指令只支持c指令

2.加上c后会直接自动执行程序，不如手动输入c灵活

任务三：

在main函数之前就会被杀死，所以在premain打断点。然后发现premain调用了magi\_init。在libmagi.so中找到magi\_init函数反编译。

关键在于monitor函数。在monitor函数中，创建子进程，并且子进程会尝试通过ptrace来检测自身是否被调试。

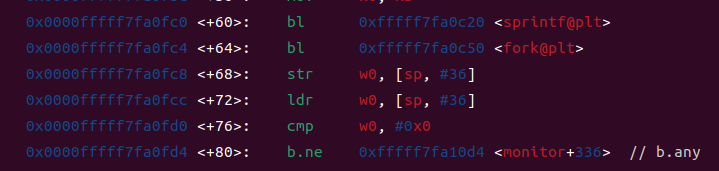


通过puts的内容“Parent have already been killed”可以推测p==0那么就会杀死父进程，所以不能跳进去。

具体来说， fork() 函数创建一个子进程，并且返回子进程的PID，通过改变p能够改变了父进程在控制子进程上的逻辑，从而避免了其中的子进程退出逻辑。

类似于ppt9

所以不能跳进去子进程



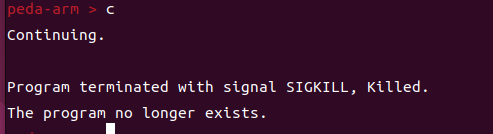
在这里可以看到在0xfffff7fa0fd0打上断点，把w0（即x0）改为非0的值，直接set $w0=1就可以绕过反调试机制。



绕过后不会再触发kill机制

如果运气够好可以看到循环跳出来的字符/tmp/media，和上一个任务一样

未绕过反调试机制的情况：



显然触发了kill机制

这个反调试机制的关键在于子进程调用ptrace(PTRACE\_TRACEME, 0, 0, 0)允许自身被调试，然后通过检查TracerPid字段来确认是否有调试器正在跟踪。

如果发现调试器存在，会发送信号给父进程，父进程可以根据这个信息采取相应的行动。