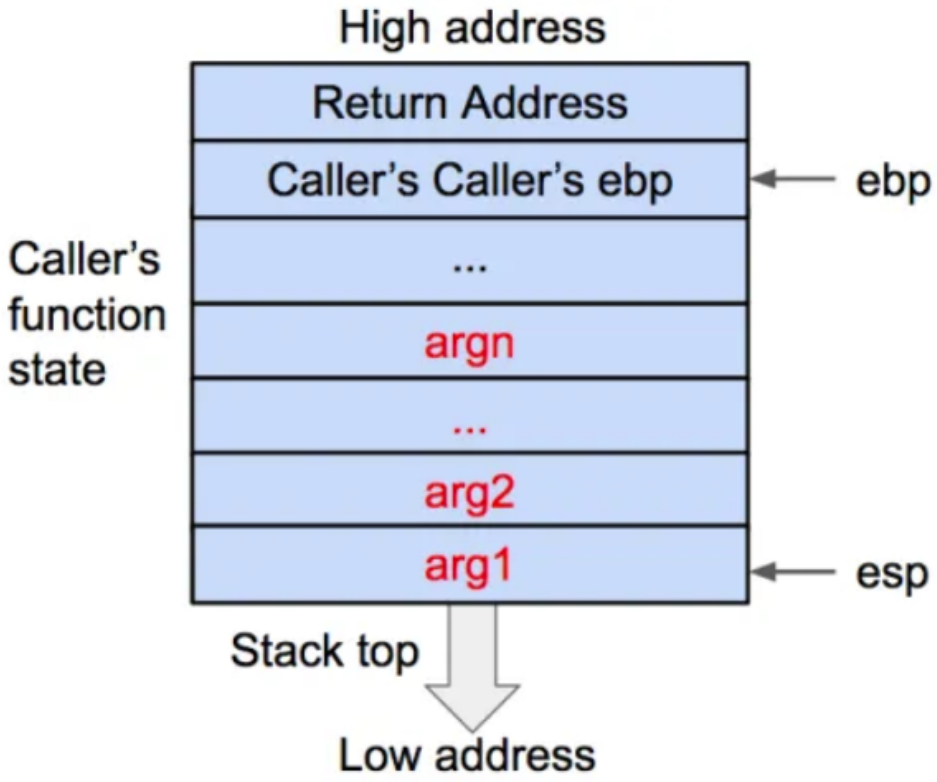
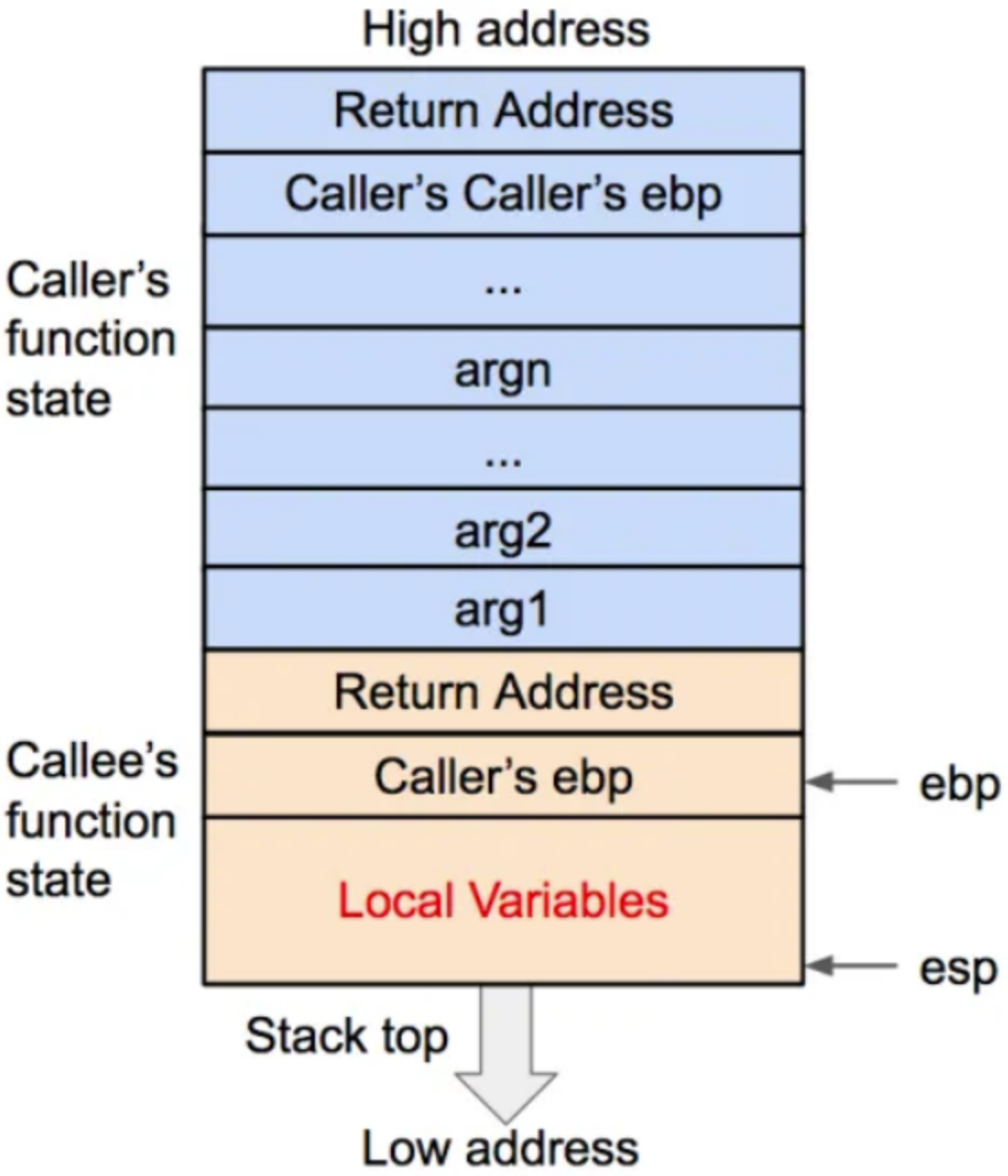
rop

回顾一下x86架构的函数调用过程。

首先将被调用函数（callee）的参数按照逆序依次压入栈内。如果被调用函数（callee）不需要参数，则没有这一步骤。这些参数仍会保存在调用函数（caller）的函数状态内，之后压入栈内的数据都会作为被调用函数（callee）的函数状态来保存。

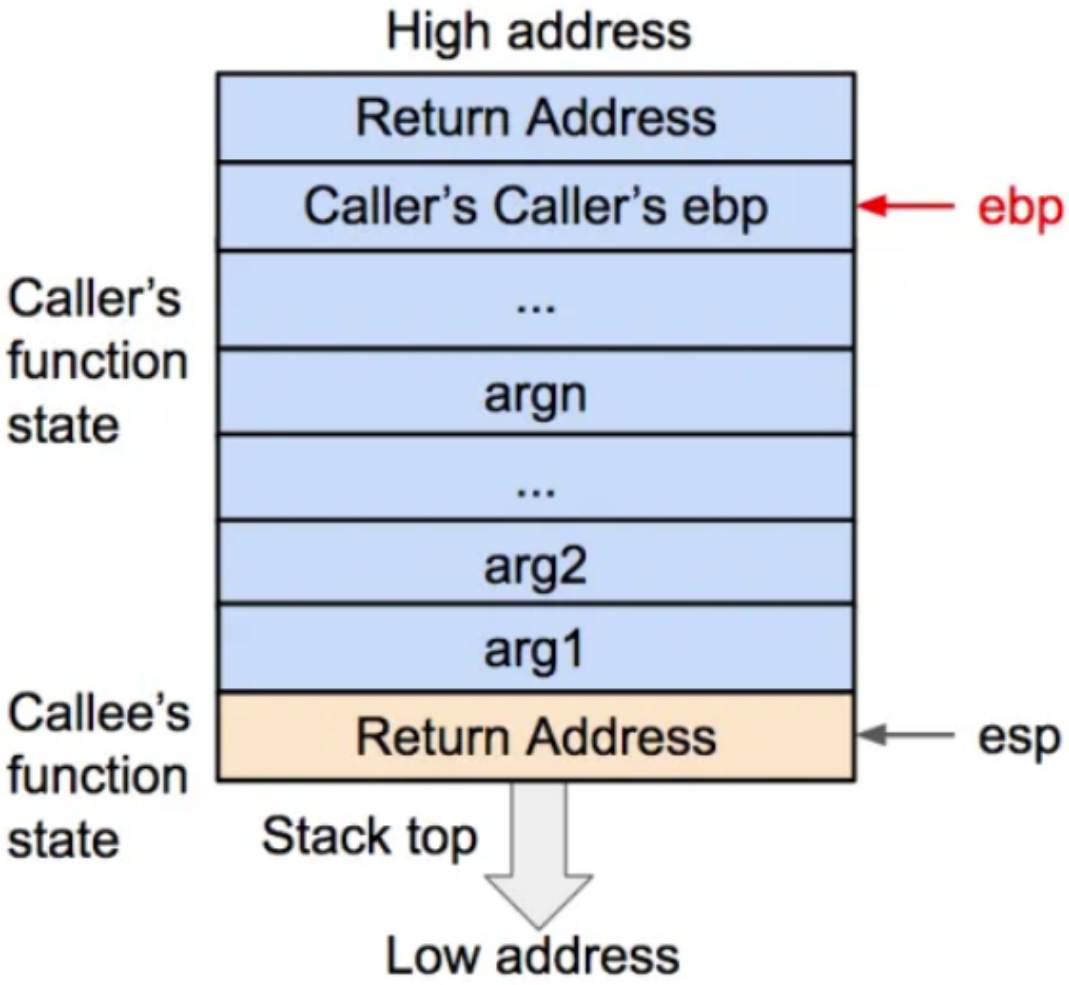


执行call指令的时候，调用函数（caller）会把函数返回地址压栈。跳转到被调函数（callee）后，先把ebp压栈，然后为被调函数的本地变量创建空间。

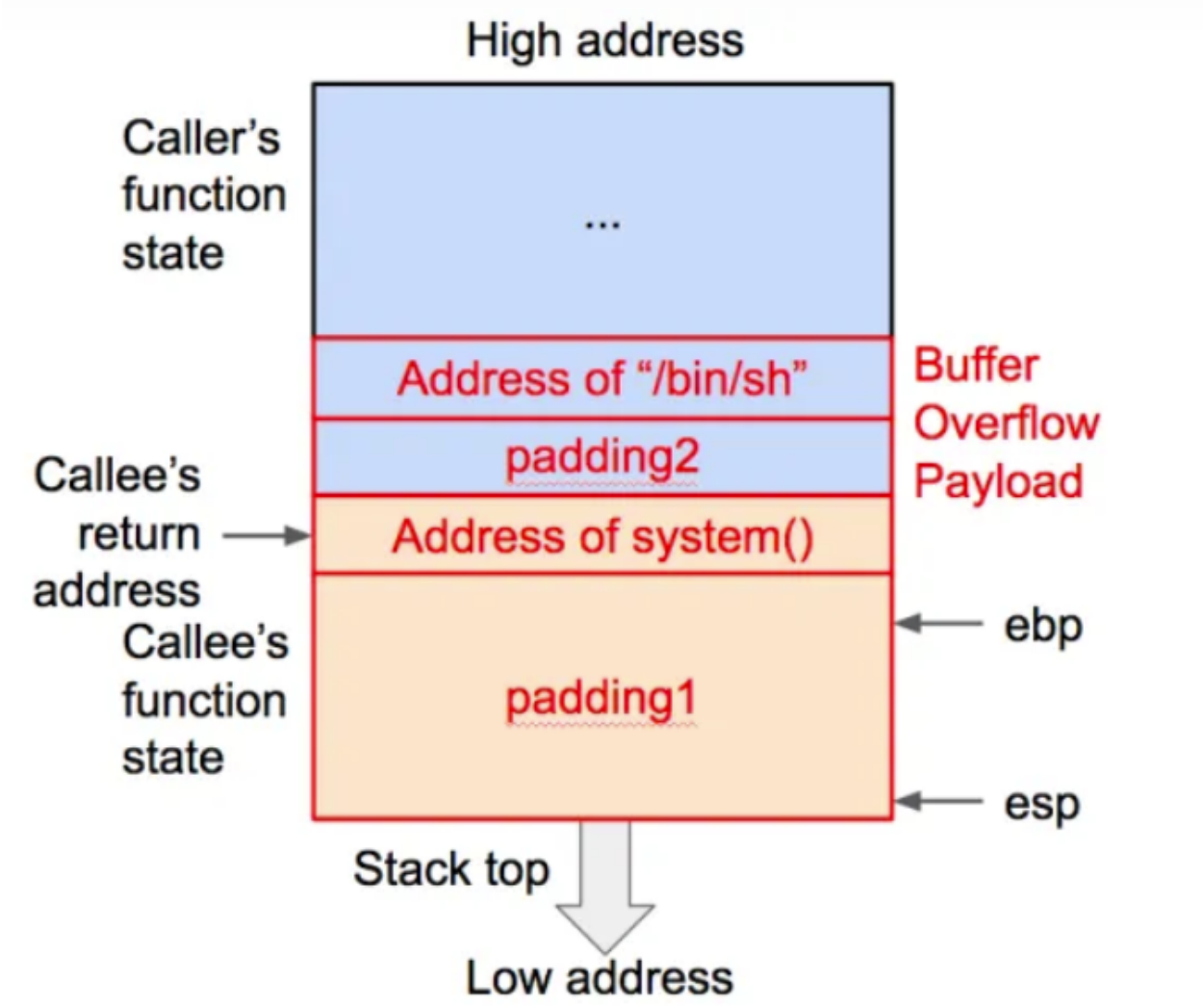


函数返回过程相反

esp会直接指向栈上存放caller ebp的内存位置，将caller栈基址恢复。



之后执行ret指令，跳转到返回地址的代码执行。攻击者就是要想办法修改返回地址劫持控制流。ret2libc就是修改返回地址，让其指向libc中的函数。



库函数在链接库中的偏移可以通过readelf查看

```
test@2-9:~$ readelf --syms /lib/x86_64-linux-gnu/libc.so.6 | grep system
   233: 0000000000159c50    99 FUNC    GLOBAL DEFAULT   13 svcerr_systemerr@@GLIBC_2.2.5
   609: 000000000004f420    45 FUNC    GLOBAL DEFAULT   13 __libc_system@@GLIBC_PRIVATE
  1406: 000000000004f420    45 FUNC    WEAK   DEFAULT   13 system@@GLIBC_2.2.5
```

库函数中存在的字符串可以通过strings查看

```
test@2-9:~$ strings -td /lib/x86_64-linux-gnu/libc.so.6 | grep /bin/sh
1785224 /bin/sh
```

x86下如果想调用多个库函数（如open, read, write），由于参数都通过栈传递，如果只一次ret，payload不好部署，参数会全部混在一起，可以考虑多次ret，每次执行完一个库函数都返回到main函数重新输入新的payload完成利用。