Linux pwn入门教程之环境配置

转载

程序员大咖 于 2018-07-13 10:24:00 发布 2108 收藏 4

前言

作为一个毕业一年多的辣鸡CTF选手，一直苦于pwn题目的入门难，入了门更难的问题。本来网上关于pwn的资料就比较零散，而且经常会碰到师傅们堪比解题过程略的writeup和没有注释，存在大量硬编码偏移的脚本，还有练习题目难找，调试环境难搭建，GDB没有IDA好操作等等问题。作为一个老萌新(雾)，决定依据Atum师傅在i春秋上的pwn入门课程中的技术分类，结合近几年赛事中出现的一些题目和文章整理出一份自己心目中相对完整的Linux pwn教程。

本系列教程仅针对i386/amd64下的Linux pwn常见的pwn手法，如栈，堆，整数溢出，格式化字符串，条件竞争等进行介绍。为了方便和我一样的萌新们进行学习，所有环境都会封装在docker镜像当中，并提供调试用的教学程序，来自历年赛事的原题和带有注释的python脚本。教程欢迎各位师傅吐槽，若对题目和脚本的使用有不妥之处，会在当事师傅反馈之后致歉并应要求进行处理。

docker容器的使用与简单操作

在搭建环境之前我们需要准备一个装有docker的64位Linux系统，内核版本高于3.10(可以通过uname -r查看)，可以运行在实体机或者是虚拟机中。关于docker的安装与启动此处不再赘述，读者可以根据自己的Linux发行版本自行搜索。此处提供两个链接，供Ubuntu和Kali使用者参考：

Kali：《kali Rolling安装docker》http://www.cnblogs.com/Roachs/p/6308896.html
Ubuntu：《Ubuntu 16.04安装Docker》http://blog.csdn.net/qq_27818541/article/details/73647797

在成功安装了docker并验证其可用性后，我们就可以定制自己的实验用容器了。这部分内容可以在各个地方找到教程，且与pwn的学习不相关，此处不再赘述。为了方便实验，我把实验环境打包成了几个容器快照，可以直接导入成镜像使用。

以ubuntu.17.04.amd64为例，导入的命令为

cat ubuntu.17.04.amd64 | docker import - ubuntu/17.04.amd64

```
root@kali:~# cat ubuntu.17.04.amd64 | docker import - ubuntu/17.04.amd64
sha256:876674af8eed40c5cef9693d7335f9b35a11dae6838105aa992f1429bda75b40
```

导入成功后使用命令docker images会看到镜像仓库中出现了一个新的镜像。

```
root@kali:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu/17.04.amd64  latest              876674af8eed        16 seconds ago      675MB
```

运行docker run -it -p 23946:23946 ubuntu/17.04.amd64 /bin/bash

就可以以这个镜像创建一个容器，开启一个shell，并且将IDA调试服务器监听的23946端口转发到本地的23946端口。

```
root@kali:~# docker run -it -p 23946:23946 ubuntu/17.04.amd64 /bin/bash
root@a8607834dacd:/# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
```

通过命令docker container ls -a 我们发现容器列表里多了一个刚刚创建的容器，并且被赋予了一个随机的名字，在我的实验中它是nostalgic_raman。

我们可以通过命令

docker container rename nostalgic_raman ubuntu.17.04.amd64

把这个容器重命名为ubuntu.17.04.amd64或者其他你认为合适的名字。

```
root@kali:~# docker container rename nostalgic_raman ubuntu.17.04.amd64
root@kali:~# docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
NAMES
a8607834dacd        ubuntu/17.04.amd64  "/bin/bash"         4 minutes ago       Up About a minute   0.0.0.0:23946->23946/tcp
ubuntu.17.04.amd64
```

使用

docker exec -it ubuntu.17.04.amd64 /bin/bash

我们可以打开目标容器的一个新的bash shell。这使得我们在后续的调试中可以在容器中启动IDA调试服务器并用socat部署pwn题目。

```
root@kali:~# docker exec -it ubuntu.17.04.amd64 /bin/bash
root@658a2deff87f:/# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
```

此外，可以使用docker container cp命令在docker容器内外双向传输文件等等。需要注意的是，对容器的各种操作需要在容器运行时进行，若容器尚未运行(运行docker container ls未显示对应容器)，需使用命令docker start运行对应容器。此外，若同时运行多个容器，为了避免端口冲突，在启动容器时，可以将命令docker run -it -p 23946:23946 ubuntu/17.04.amd64 /bin/bash 中的第一个端口号23946改为其他数字。

IDA的简单使用及远程调试配置

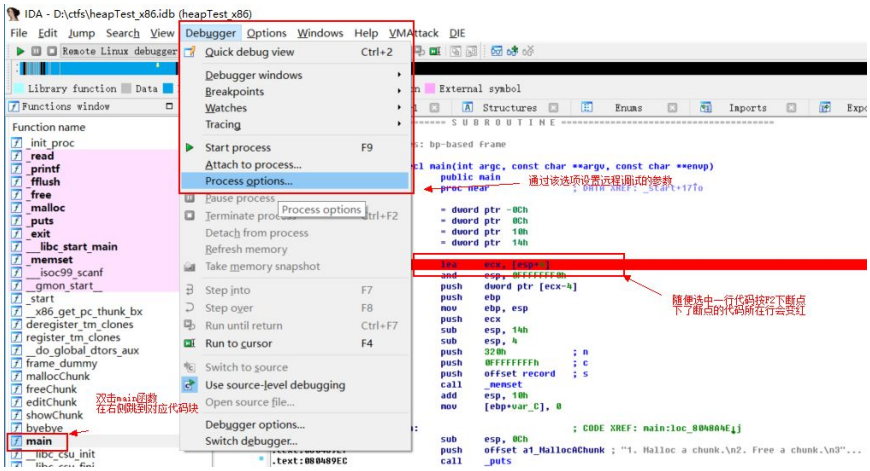
成功搭建了docker环境之后，我们接下来熟悉一下IDA和IDA的远程调试环境搭建。首先我们在IDA所在的文件夹的dbgsrv文件夹下找到需要的调试服务器linux_server(32位)和linux_serverx64(64位)并复制到kali中。

然后使用命令

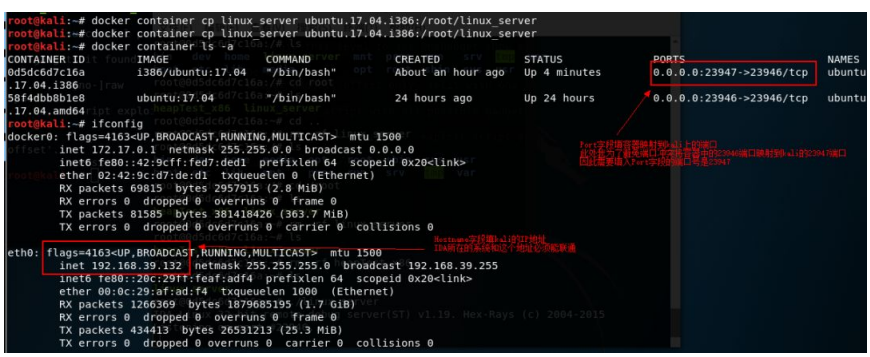
dockercontainercplinux_server ubuntu.17.04.i386:/root/linux_server

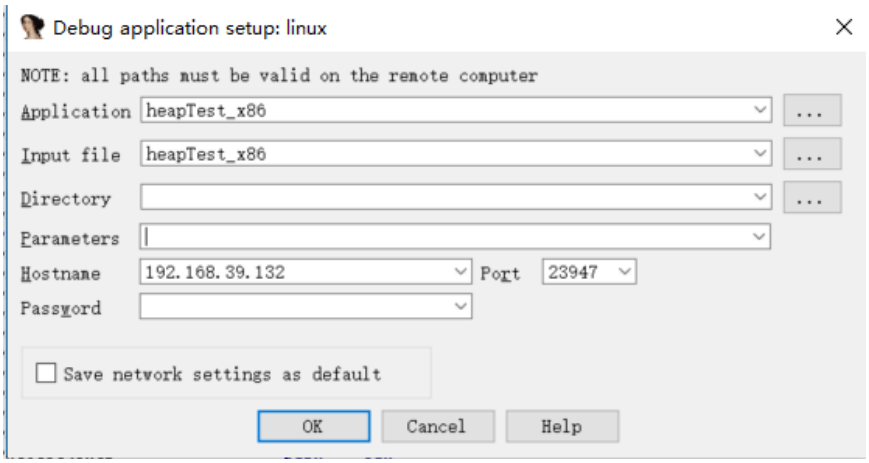
将linux_server复制到32位容器中的/root目录下。此时我们登录容器可以看到linux_server，运行该server会提示正在监听23946端口。

接着我们打开32位的ida，载入一个后面会用于演示堆漏洞的程序heapTest_x86，在左侧的Functions window中找到main函数，随便挑一行代码按F2下一个断点。然后通过Debugger->Process options...打开选项窗口设置远程调试选项。



在弹出的选项窗口中配置Hostname为kali的ip地址，Port为容器映射到kali中的端口。

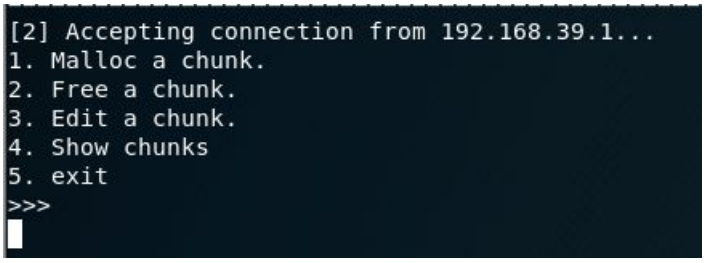




填好后点击OK，按快捷键F9运行程序。若连接正常可能提示Input file is missing:xxxxx，一路OK就行，IDA会将被调试的文件复制到服务器所在目录下，然后汇编代码所在窗口背景会变成浅蓝色并且窗口布局发生变化。若IDA僵死一段时间后跳出Warning窗口，则需要检查IDA所在机器与kali是否能ping通，容器对应端口是否映射，参数是否填错等问题。

调试器连接成功后我们就可以使用各种快捷键对目标程序进行调试，常用的快捷键有 下断点/取消断点 F2，运行程序F9，单步跨过函数F8，单步进入函数F7，运行到选中位置F4等等。在调试模式下主要使用到的窗口有汇编窗口 IDA View-EIP，寄存器窗口General registers，栈窗口Stack view，内存窗口Hex View，系统日志窗口Output window等。

切回到kali，我们会看到随着程序运行，运行调试服务器的shell窗口会显示出新的内容



当IDA中的程序执行完

call ___isoc99_scanf

或者类似的等待输入的指令后会陷入阻塞状态，F4，F7，F8，F9等和运行相关的快捷键都不生效。此时我们可以在shell中输入内容，IDA中的程序即可恢复执行。

使用pwntools和IDA调试程序

在上一节中我们尝试了使用IDA配置远程调试，但是在调试中我们可能会有一些特殊的需求，比如自动化完成一些操作或者向程序传递一些包含不可见字符的地址，如P(0x08048350)。这个时候我们就需要使用脚本来完成此类操作。我们选用的是著名的python库pwntools。 pwntools库可以使用pip进行安装，其官方文档地址为http://docs.pwntools.com/en/stable/ 。在本节中我们将使用pwntools和IDA配合调试程序。

首先我们在kali中安装pwntools，安装完成后输入python进入python环境，使用from pwn import * 导入pwntools库。

```
root@kali:~# python
Python 2.7.14 (default, Sep 17 2017, 18:50:44)
[GCC 7.2.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from pwn import *
>>>
```

使用docker exec在32位的容器中新开一个bash shell，跳转到heapTest_x86所在目录/root，查看容器的IP地址，然后执行命令

socattcp-listen:10001,reuseaddr,fork EXEC:./heapTest_x86,pty,raw,echo=0

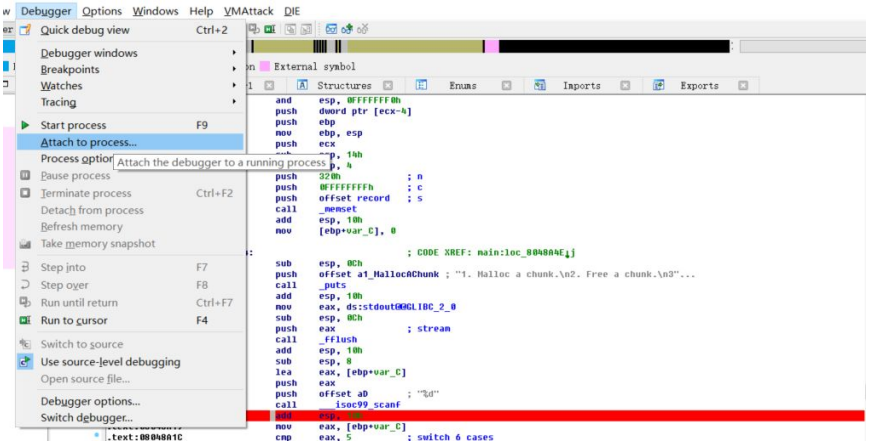
将heapTest_x86的IO转发到10001端口上。

```
root@dd241a9ea512: ~
File  Edit  View  Search  Terminal  Help
root@dd241a9ea512:~# socat tcp-listen:10001,reuseaddr,fork EXEC:./heapTest_x86,p
ty,raw,echo=0
```

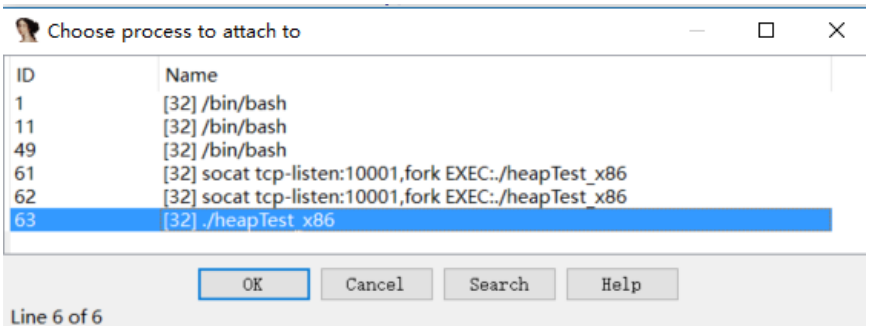
我们可以看到我的容器中的IP地址是172.17.0.2。回到python中，使用io = remote("172.17.0.2", 10001)打开与heapTest_x86的连接。

```
>>> io = remote("172.17.0.3", 10001)
[x] Opening connection to 172.17.0.3 on port 10001
[x] Opening connection to 172.17.0.3 on port 10001: Trying 172.17.0.3
[+] Opening connection to 172.17.0.3 on port 10001: Done
```

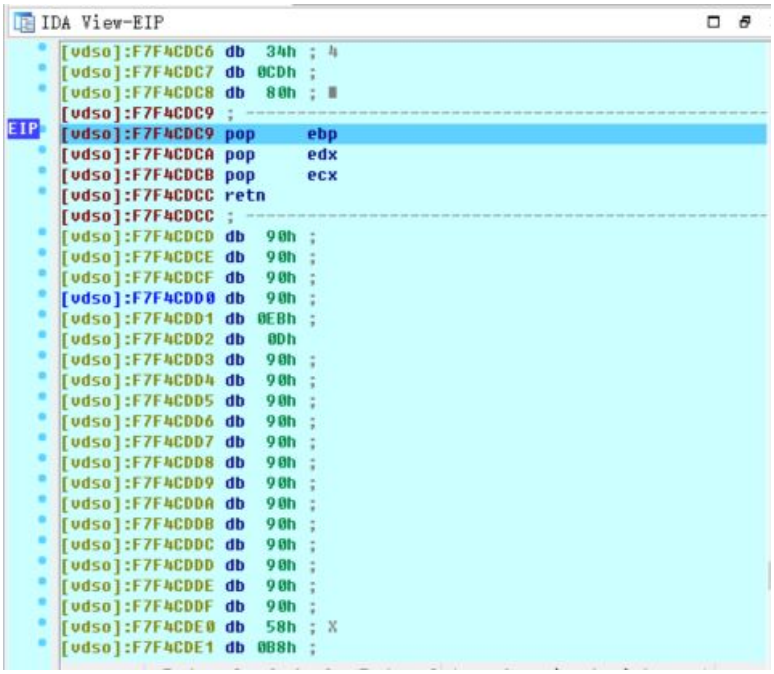
这个时候我们返回到IDA中设置断点。需要注意的是此时heapTest_x86已经开始运行，我们的目标是附加到其运行的进程上，所以我们需要把断点设置在call ___isoc99_scanf等等待输入的指令运行顺序之后，否则由于计算机的运行速度，我们的断点将会因为已经目标指令已经执行完而失效，达不到断下来的效果。



选择Debugger->Attach to process...，附加到./heapTest_x86的进程上。

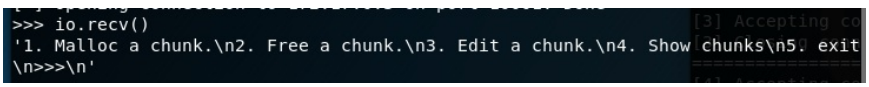


此时EIP将指向vdso中的pop ebp指令上。



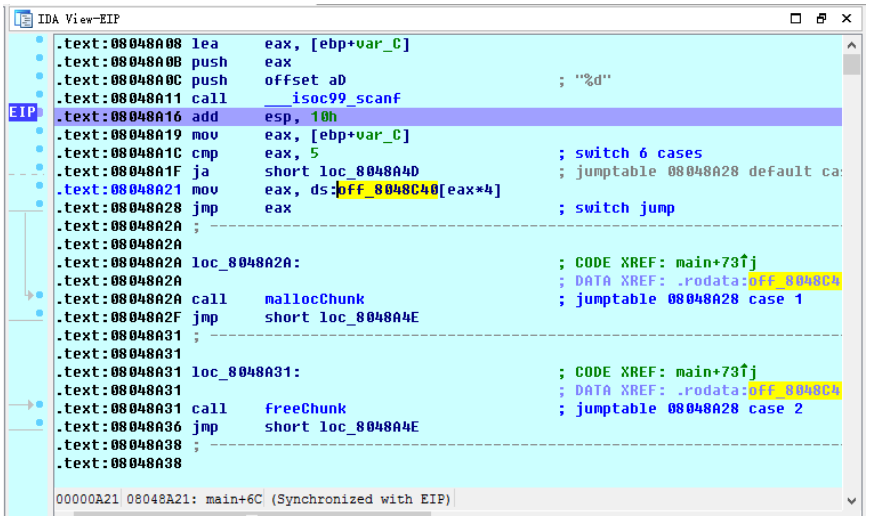
这几行指令实际上是执行完sys_read后的指令，此处我们不需要关心它，直接按F9，选中标志会消失。

回到python窗口，我们使用pwntools的recv/send函数族来与运行中的heapTest_x86进行交互。首先输入io.recv()，我们发现原先会在shell窗口出现的菜单被读出到python窗口里了。



同样的，我们通过io.send()也可以向这个进程传递输入。我们使用io.send('1')告诉这个进程我们要选择选项1。这个时候我们切换到IDA窗口，发现IDA还是处于挂起状态，这是为什么呢？

回想一下我们通过shell与这个进程交互的时候，输入选项后需要按回车键以“告诉”这个进程我们的输入结束了。那么在这里我们同样需要再发送一个回车，所以我们再执行io.send(''），切换到IDA窗口就会发现EIP停在了熟悉的程序领空。这时候我们再使用IDA的快捷键就可以进行调试，随心所欲地观察进程的内存，栈，寄存器等的状态了。当然，我们也可以直接使用io.sendline()，就可以直接在输入的结尾自动加上"了。



在上图的状态中，我们在python中再次输入io.recv()，发现并没有读取到输出，并且python处于阻塞状态。这是因为程序此时没有输出可读取。我们在IDA中按F8到call mallocChunk一行，此时按F7进入函数，在函数中运行到call _fflush的下一行，就会发现python的阻塞状态解除了。

当我们希望结束调试时，应该使用io.close()关闭掉这个io。否则下一次试图attach时会发现有两个./heapTest_x86进程。在IDA中按Ctrl+F2即可退出调试模式。