# Mirai分析

拿到一个样本（可执行程序）后，可以先做静态分析，然后结合动态调试，复现样本的恶意行为。这里列举了一些文件测、网络侧的行为（有遗漏可以补充），挑选了其中部分主机测、网络侧的行为进行分析及复现，仅供参考。

如果是想要搭环境，可以直接看第3部分；如果是分析Mirai，看前两个部分就行——这里选了文件隐藏和DDoS攻击进行复现，将会讲述本人在动态调试的过程中，遇到的一些问题，如何解决这些问题，如何结合源码（静态分析）复现恶意行为等。

## 1 主机侧（文件侧）

主机侧指的是恶意程序在受感染设备上进行的一些本地操作。

### 1.1 隐藏

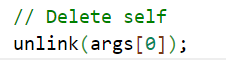
这里把隐藏行为分成**进程隐藏**和**文件隐藏**。

#### 1.1.1 文件隐藏

在类Unix系统（如Linux、macOS）中，使用unlink(args[0])来删除正在执行的程序文件时，文件会从文件系统的目录中被删除，但文件的内容仍然保留在硬盘上，直到所有打开的文件描述符（包括正在执行的程序所用的描述符）都关闭为止。

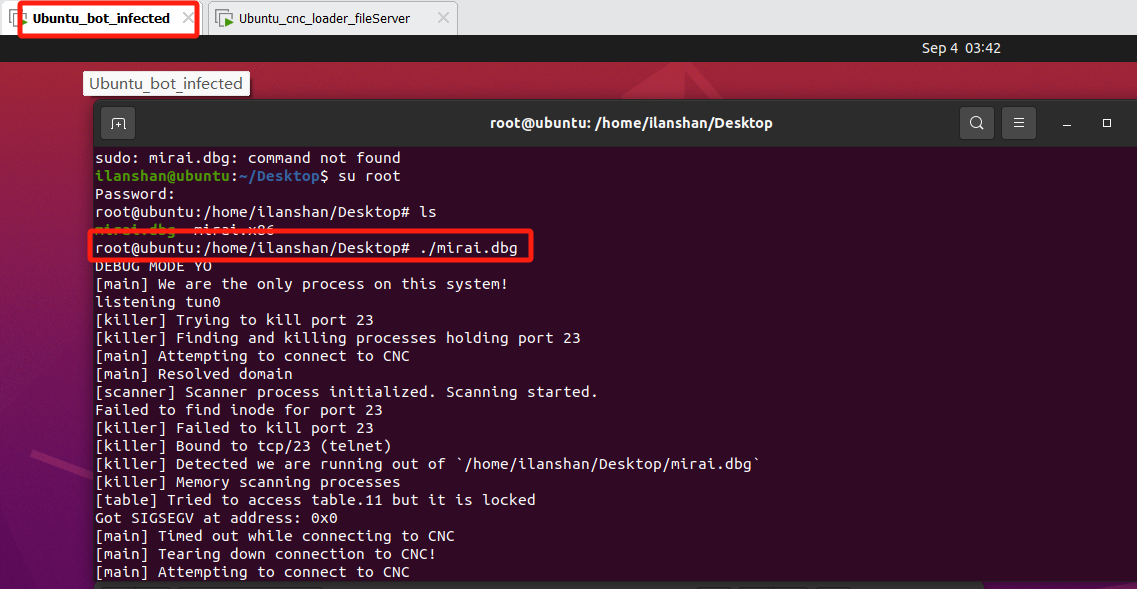
在args[0]一般是指向程序的名称，如执行”./myprogram arg1 arg2”，那么”args[0]”将指向”./myprogram”。

因此，如下图所示，在程序执行过程中，删掉自身文件这个行为是不会干扰程序正常运行的。



（该图来自.../mirai/bot/main.c的第61行左右）

动调情况如下，先把bot程序在虚拟机Ubuntu\_bot\_infected上跑起来。



然后杀死该进程，观察文件是否仍然存在（可以通过”kill -9 <pid>”选择杀死指定pid的进程）。在把进程杀完后，这里发现文件mirai.dbg还在，为什么还在呢？不妨再看看源码。



（该图来自.../mirai/bot/main.c第56行左右）

这里有个条件宏，只有当未定义”DEBUG”时，下面的代码才会正常执行，而我们编译的代码很有可能是定义了”DEBUG”的。在查找过目录bot下各个文件后，未找到定义”DEBUG”的位置，说明定义”DEBUG”的位置可能来自于编译指令。

还记得编译mirai.dbg的脚本指令是”sudo ./build.sh debug telnet”，不妨再去看看”build.sh”脚本，在脚本的第49行，可以看到这么一条指令，这一条便是生成mirai.dbg的编译指令。



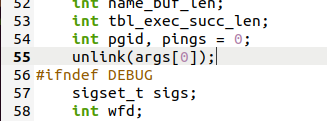
（该图来自.../mirai/build.sh的第49行左右）

其中的-DDEBUG便是添加宏定义DEBUG，因此，可以修改脚本（去掉-DDEBUG），重新编译一个mirai.dbg；或者根据编译选项，选择编译一个release版本，release版也没有定义”DEBUG”；或者将unlink这行代码移除条件宏范围内。



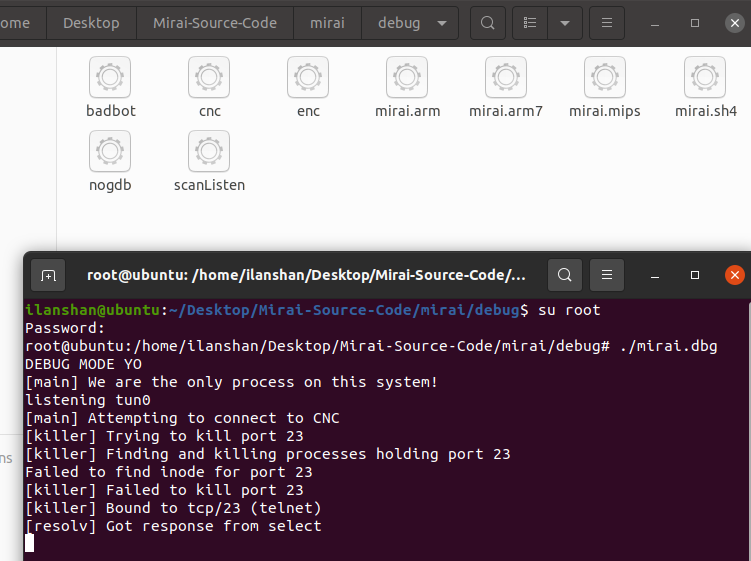
（该图来自.../mirai/build.sh）

我在虚拟机Ubuntu\_bot上装了一个mirai的编译环境，unlink(args[0])这段代码移出DEBUG的条件宏范围内，然后重新编译并用root权限执行mirai.dbg。（用&作为结尾可以显示进程的pid并令进程在后台运行）



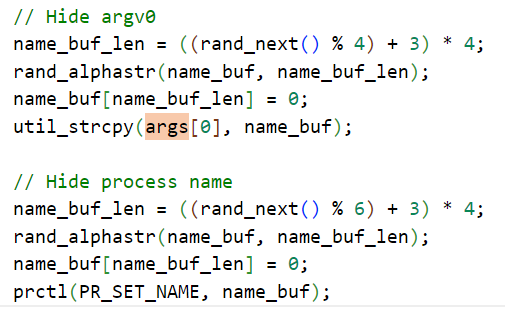
（该图来自.../mirai/bot/main.c第55行左右）

如下图所示，在程序执行后，目录debug下的mirai.dbg消失了。



#### 1.1.2 进程隐藏

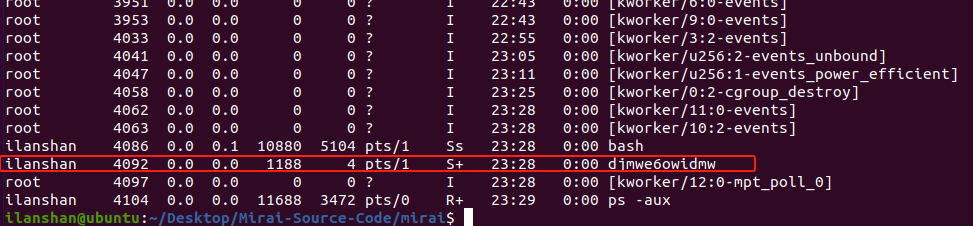
在linux系统中，常见的用来查看进程的指令有ps、top等，mirai在进程隐藏方面，先是生成随机字符串，然后对进程名进行修改。



可以在bot执行时，另开一个终端，通过ps查看是否有名为mirai.dbg的进程，多半是找不到，因为名字被替换成了随机字符串。

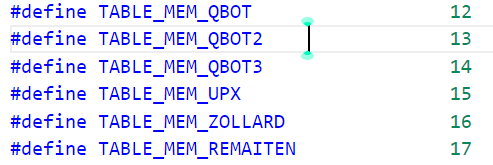


但因为其进程名是随机的，还是很好辨别的。

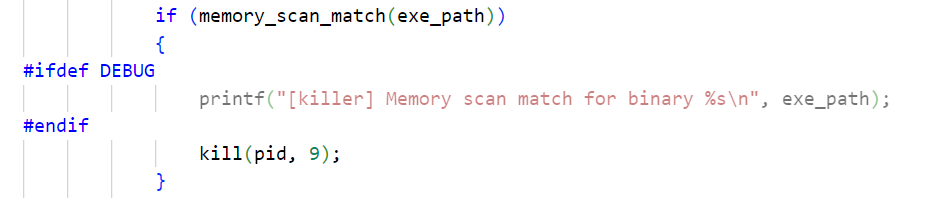


### 1.2 杀死同类进程

Mirai会通过文件扫描，定位并除去其它的僵尸网络的进程，下图是mirai所针对的几个恶意家族对象，如qbot、zollard等。



通过扫描正在运行的进程，找到进程对应程序的源文件，然后将该文件读入内存，进行相关特征匹配，匹配正确就将该进程杀死。



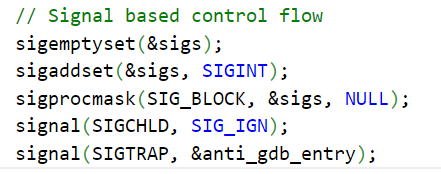
### 1.3 反调试

INT3是x86汇编语言中的一个软件中断指令，用于生成一个类型为3的中断。在调试器中，INT3指令通常被用来设置断点。当程序执行到INT3指令时，它会触发一个中断，告诉操作系统有一个断点被命中了。

而SIGTRAP是一个信号，在POSIX兼容的操作系统中用于表示陷阱信号，在Linux和其他UNIX-like系统中，当程序遇到异常情况，如断点命中，SIGTRAP信号会被发送给该程序。

在调试器（如gdb）中，当我们设置一个断点，调试器会在目标程序的内存中将要中断的指令替换为 INT3 指令。当程序执行到这个位置时，INT3指令会被执行，触发一个类型为3的中断。操作系统会捕获这个中断，并将其转换为SIGTRAP信号发送给调试器，调试器保留了中断处原先的指令，把INT3替换成原来的指令，以便在恢复程序执行时不干扰程序的正常运行。

在mirai源码中，有这么几行代码，如下图所示。

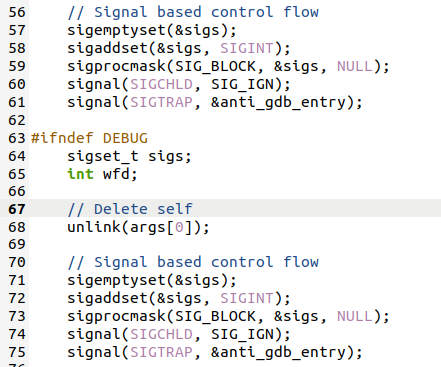


（该图来自.../mirai/bot/main.c第68行左右）

signal(SIGTRAP, &anti\_gdb\_entry)的作用是，当触发了SIGTRAP信号，进程会可能会“接管”对该信号的处理，导致调试器无法正常捕获和处理断点，一定程度上阻止调试行为。

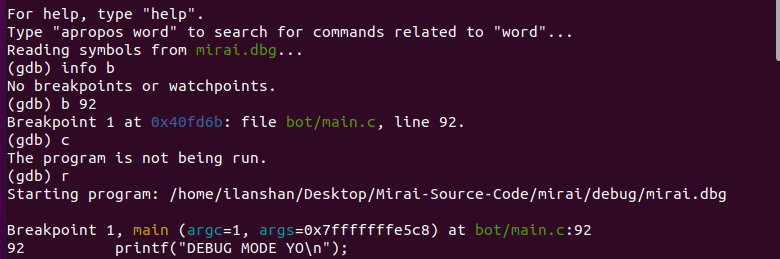
为了动调这个功能，要先安装一个gdb，gdb的使用方法可以网上查。

上面几行代码都处于条件宏中，需要把他们移出去。



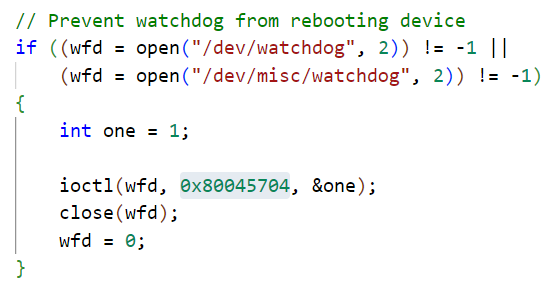
（该图来自.../mirai/bot/main.c）

然后保存，重新编译，利用gdb进行调试，在第61行之后，第92行下了一个断点，结果操作系统仍然把控制权给到了调试器，并没有转去执行anti\_gdb\_entry，说明这个反调功能有待提高。



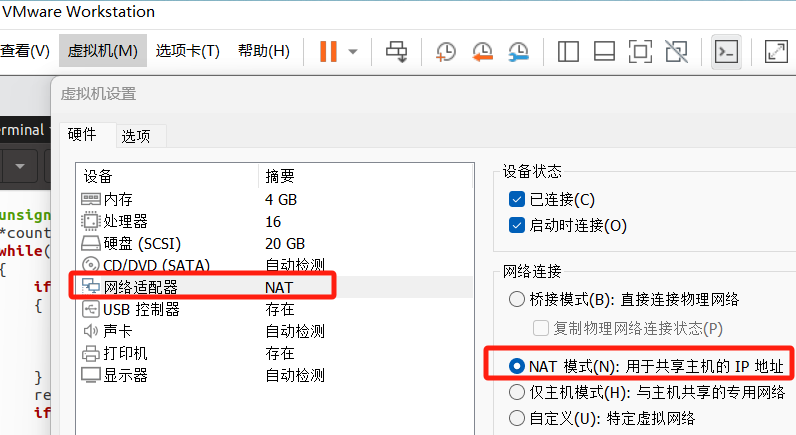
### 1.4 关闭watchdog

看门狗（watchdog）一种程序，它在后台运行并监控系统或应用程序的状态。如果软件看门狗检测到系统或应用程序在一定时间内没有响应，它会采取措施，比如重启服务。为了防止设备做出反应，mirai利用函数ioctl对看门狗程序进行关闭。

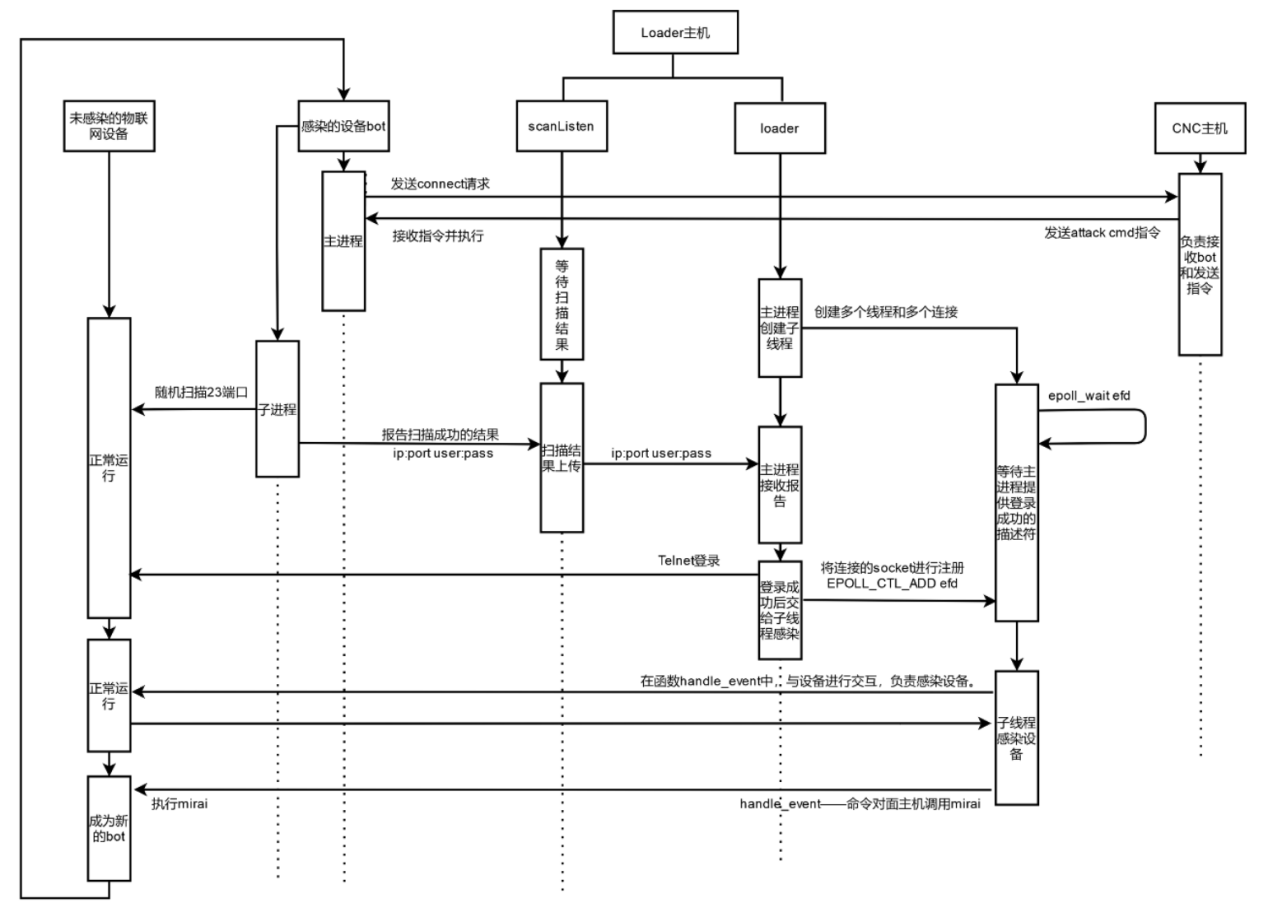


## 2 网络侧

网络侧指的是受感染设备在恶意程序控制下进行的一些网络通信。为了能够保证bot与cnc的连接，需要将虚拟机放在同一个局域网内。为了保证处于同一网段，可以在VMware Workstation的选项栏中，找到”虚拟机”->”设置”->”网络适配器”，修改成NAT模式。



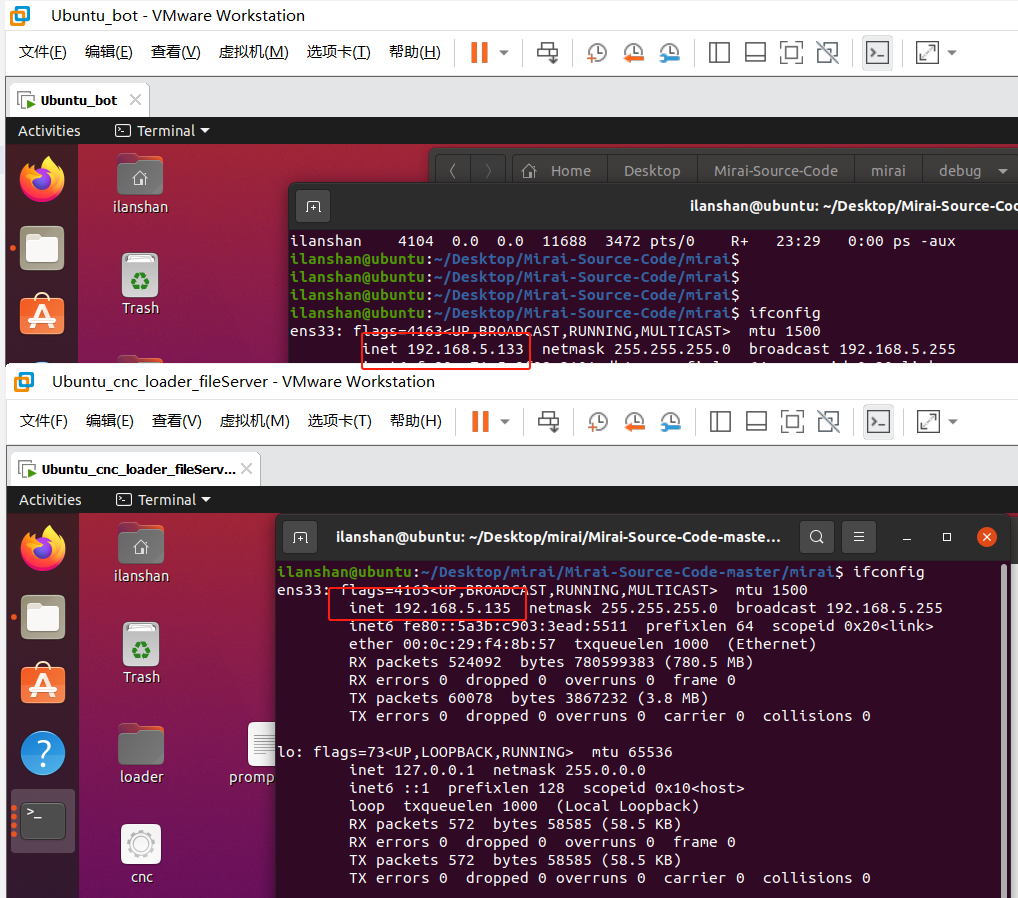
Mirai的网络框架如下图所示。



### 2.1 bot与cnc通信

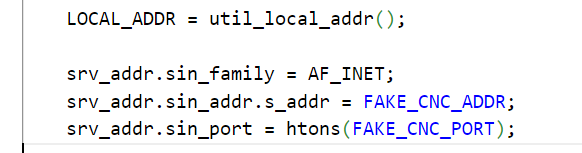
我这边使用的是Vmware Workstation，以这个软件为例，如何实现bot与cnc进行通信。在VMware Workstation上开了2台虚拟机，一台名Ubuntu\_bot，作为bot端；一台名Ubuntu\_cnc\_loader\_fileServer，作为cnc端。

本人将用来做实验的虚拟机的网络连接都设置为NAT模式，以保证处于一个局域网内。可以通过ifconfig查看，bot的ip为192.168.5.133，而cnc的ip为192.168.5.135，而子网掩码是24位，都属于局域网192.168.5.X。



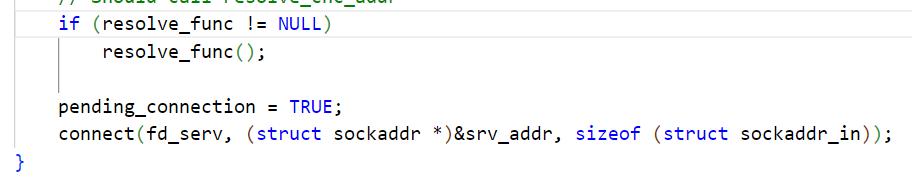
其次，要了解bot是如何联系上cnc端的，**以下是分析（静调和动调都有）。**

阅读代码可以知道，srv\_addr是负责存储cnc服务器的全局变量，第一次出现在.../mirai/bot/main.c第105行左右，这里是给srv\_addr赋了一个硬编码的假IP地址。



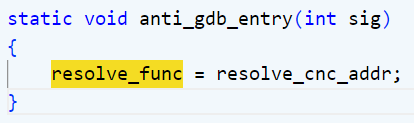
再来看看实在哪里真正利用这个srv\_addr进行通信的。

在.../mirai/bot/main.c第179行左右，有一个函数establish\_connection，它负责令bot向cnc发送连接请求，在函数establish\_connection中，会调用resolve\_func()。

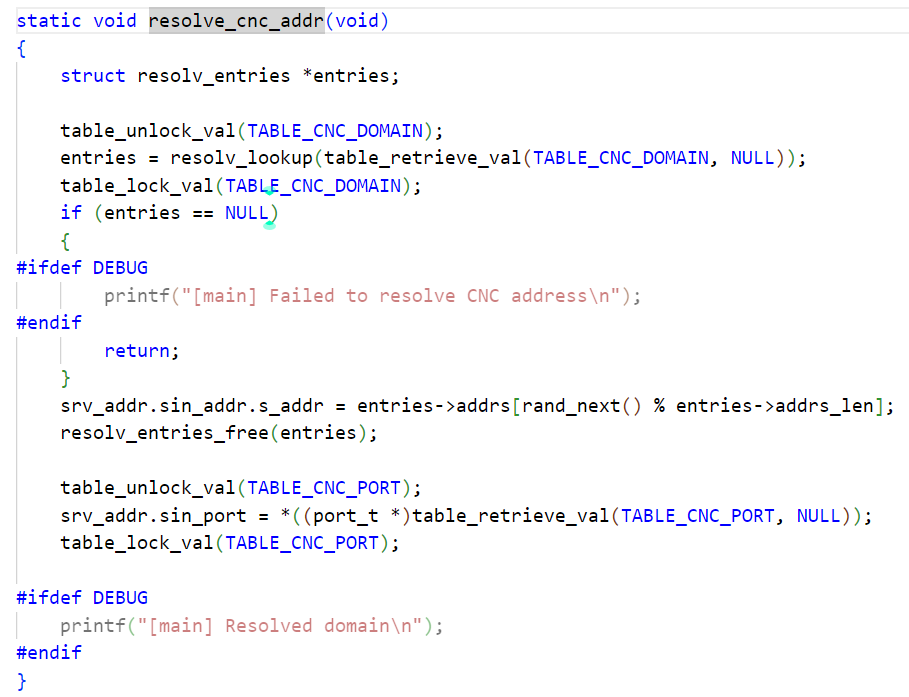


而函数resolve\_func()会将cnc的域名转换成ip，如果追踪函数指针resolve\_func，会发现它在声明定义时指向了函数util\_local\_addr，函数util\_local\_addr会获取本机的ip地址，而不是cnc的ip地址。

除开resolve\_func的声明定义，它还有被引用的地方，被赋值指向函数resolve\_cnc\_addr。



在函数resolve\_cnc\_addr中，resolv\_lookup会负责dns解析，将cnc的域名转换成ip地址。



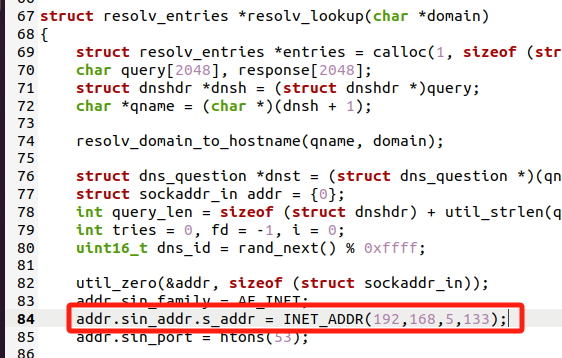
TABLE\_CNC\_DOMAIN对应的是硬编码的加密字节串，将它通过table\_unlock\_val解密后，就可以通过table\_retrieve\_val提取，获得cnc域名。

TABLE\_CNC\_DOMAIN是我们在搭环境、编译bot之前修改过的，不记得的可以看看3.3章。

对上述分析做一个总结，bot是通过域名解析，获得cnc的ip后，向cnc发起连接，而这个域名在我们之前第3.3章时修改过，是已知的。（我设置的是cnc.mirai.com）

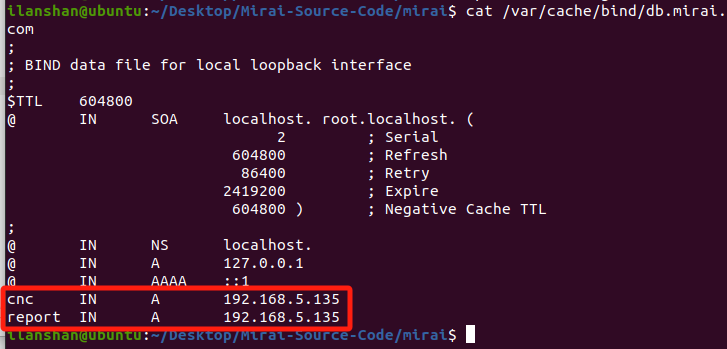
下一步便是要搭建dns服务器，这一步在网上有很多教程，搜索”ubuntu dns服务器 搭建”大概就能找到教程了，如果有同学搭建了后，发现bot仍然无法连向cnc的ip，不妨看看有没有修改.../mirai/bot/resolv.c的函数resolv\_lookup，在这个函数中硬编码了dns服务器，设置的ip为8.8.8.8，而非同学自己所配置的dns服务器的ip。

所以接下来，需要在.../mirai/bot/resolv.c中，将dns服务器修改成自己搭的dns服务器的ip，我把dns服务器建在bot虚拟机上，所以dns的ip为192.168.5.133。

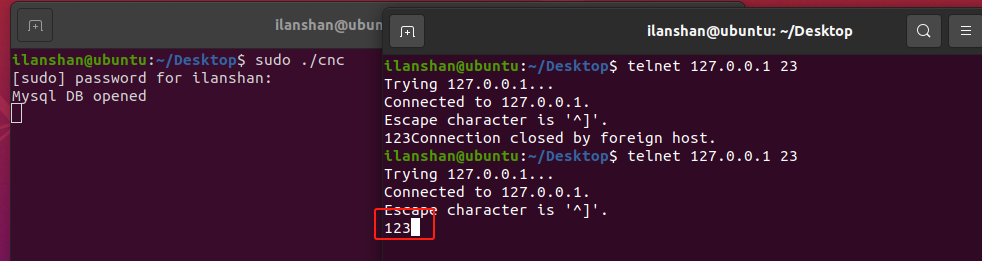


我用bind9搭建了一个dns服务器，添加了两条A记录。搭建DNS的教程如下：

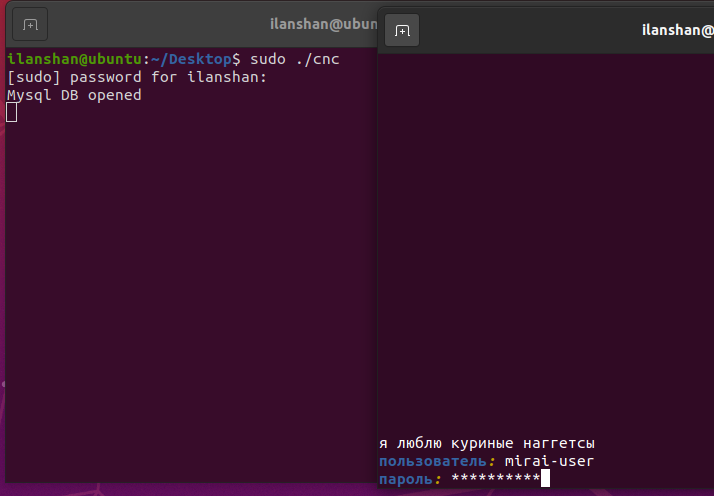
[https://www.bilibili.com/video/BV1G7411L7K5/?spm\_id\_from=333.337.search-card.all.click&vd\_source=d6d520b3115afffdc5c169cb79d7c2e8](https://www.bilibili.com/video/BV1G7411L7K5/?spm_id_from=333.337.search-card.all.click&vd_source=d6d520b3115afffdc5c169cb79d7c2e8）)



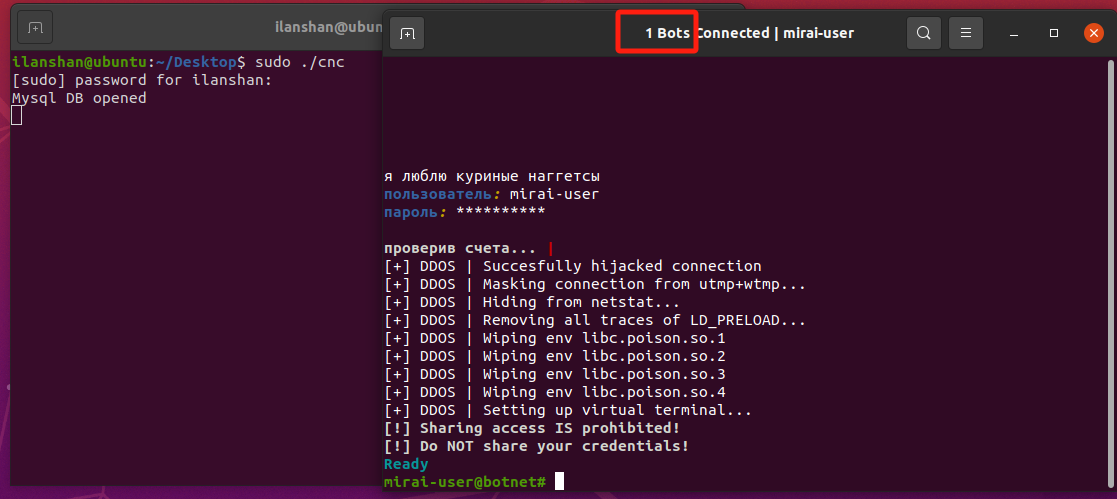
之后开启mirai.dbg，它会正确解析cnc.mirai.com和report.mirai.com的ip地址；再在虚拟机Ubuntu\_cnc\_loader\_fileServer上开启cnc，在本地使用telnet 127.0.0.1 23去登录cnc。（这里的123输错了）



然后输入mirai-user作为账号、再输入mirai-pass作为密码。（这里的账号密码是在3.4章，往数据库中插入的账号密码）



此时可以看到，已经有1个bot连上了cnc。



### 2.2 bot的telnet爆扫

bot使用有限状态机去模拟telnet的连接，尝试利用60多组账号密码去登录目标设备，这一过程包括两个步骤：可感染设备探测、可感染设备爆破。

先介绍可感染设备探测。**只有开启了telnet服务，才算是可感染设备。**下面会介绍mirai的bot是如何检测目标设备是否开启了telnet服务。

mirai使用的是syn扫描来检测目标设备是否开启了telnet端口（23端口），其扫描速度较僵尸网络QBot来说快了不少。由于telnet连接会先进行tcp三报文握手，如果目标设备的23端口是关闭的，tcp的三报文握手只可能执行第一步，即bot向目标设备发送请求连接报文；如果目标设备未关闭23端口，服务器则会回应bot，发送tcp三报文握手的第2个报文，**mirai的bot便是根据三报文握手的第2个报文来判断对方是否开启了telnet服务。**



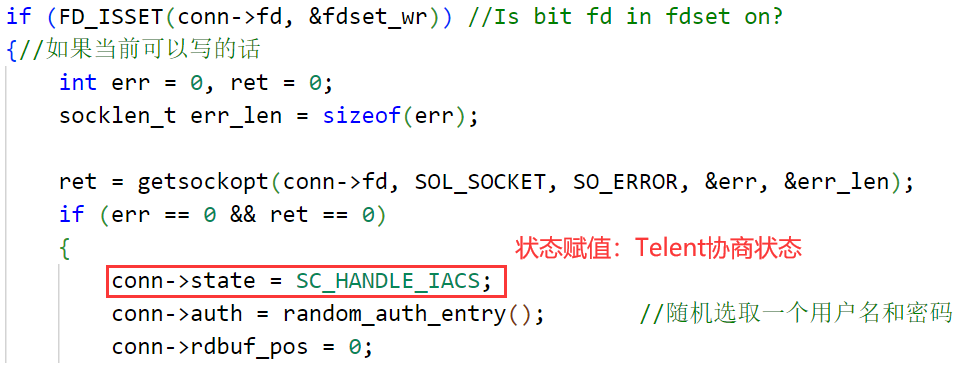
接下来是爆破阶段，在爆破阶段过程中，bot会选择利用有限状态机来模拟telnet登录过程，而在一个正常的telnet登录过程中，第一步是建立tcp连接，第二步是进行telnet协商，第三步是telnet客户端输入账号、密码进行登录。（大家可以自己试试抓telnet登录过程的包，学习一下）

第一步是建立tcp连接，与服务器（目标设备的telnet服务器）建立tcp连接的过程由函数setup\_connection实现，如下图所示。



第二步则是telnet协商，协商的过程中，客户端、服务端会讨论一些选项是否开启，如：终端类型、窗口大小等。

在bot源码中可以看到，凡是“可写”的套接字都是正处于telnet协商过程，bot会将这些可写套接字的连接状态设定成状态“SC\_HANDLE\_IACS”，这里的“IACS”指的是命令解释符。



在telnet协商过程中，bot作为telnet连接的客户端，要与服务器进行窗口大小、终端速度、远程流量控制等选项的协商。



在telnet协商过程中，bot对待服务器发来的协商报文，一共有3个回应方式。

一是，假如服务器发送的是关于窗口大小的协商，Bot将发送tmp1、tmp2两个uint8\_t的数组作回应，tmp1数组表示同意激活窗口大小的选项，而tmp2数组补充了窗口大小、位置等各个参数细节；

二是凡服务器要求bot激活某个选项，bot将全部拒绝，具体操作为：协商的选项码不变，只改变命令码，即将0xfd改成0xfc（DO改成WONT）；

三是凡服务器向bot表达——服务器端将开启某个功能，Bot将全部接收，具体操作为：协商的选项码不变，只改变命令码，即将0xfb改成0xfd，（WILL改成DO）。上述的协商过程发生在函数consume\_iacs中。



协商过程不难，但如果是利用ubuntu复现telnet爆扫，这里可能会出点问题——代码执行会迟迟停留在协商阶段，无法进入输入账号密码的阶段。

抓一轮正常telnet协商过程的包，就会发现在协商过程中，bot和服务器进行了不止一次报文协商，可能会出现两、三次协商对话。而在文件scanner.c中，telnet协商在经历了一次协商对话后便进入输入账号密码的状态，因此，情况就变成了：服务器向bot发送第二次协商的报文时，bot的连接状态早已来到“SC\_WATING\_USERNAME”，无法再处理协商报文。简单来说，bot在等服务器发送“请输入账号”的报文，而服务器在等待bot回应协商报文，双方就这么死锁了。

**下图是我对源代码的改进，允许最多进行3次协商。**



将修改后的代码重新编译后，能够进入下一阶段了——输入账号、密码阶段。在telnet协商过后，服务器会发来“请输入账号”这样类似的内容的报文，这样的报文通常在最后会以一些特殊符号作提示语来结尾，表示请用户输入账号，常见的特殊符号有冒号、美元符号、百分号等。因此，Bot其实是通过接收服务器发来的报文，判断报文中是否存在“:”、“>”、“$”、“#”、“%”、“login”、“enter”字段，以此来判断服务器是否发送了“请输入账号”的报文。

如果Bot确定当前可以发送账号，便会随机选取一组账号、密码，将其中的账号发送给服务器，然后来到输入密码的阶段。

bot程序在判断连接是否来到输入密码的阶段也是依靠提示语符号的，在这里笔者不再过多描述。在输入完密码后，Bot程序首要的任务便是要判断输入的账号、密码是否正确。mirai采用了比较巧妙的方法去验证，Bot程序在将密码传送给服务器后，接着会依次发送“enable”、“system”、“shell”、“sh”、“/bin/busybox MIRAI”字段，前4个字段是为了进入交互模式，然后在交互模式的情况下执行命令“/bin/busybox MIRAI”，如果不存在MIRAI组件，会返回内容“MIRAI: applet not found”。

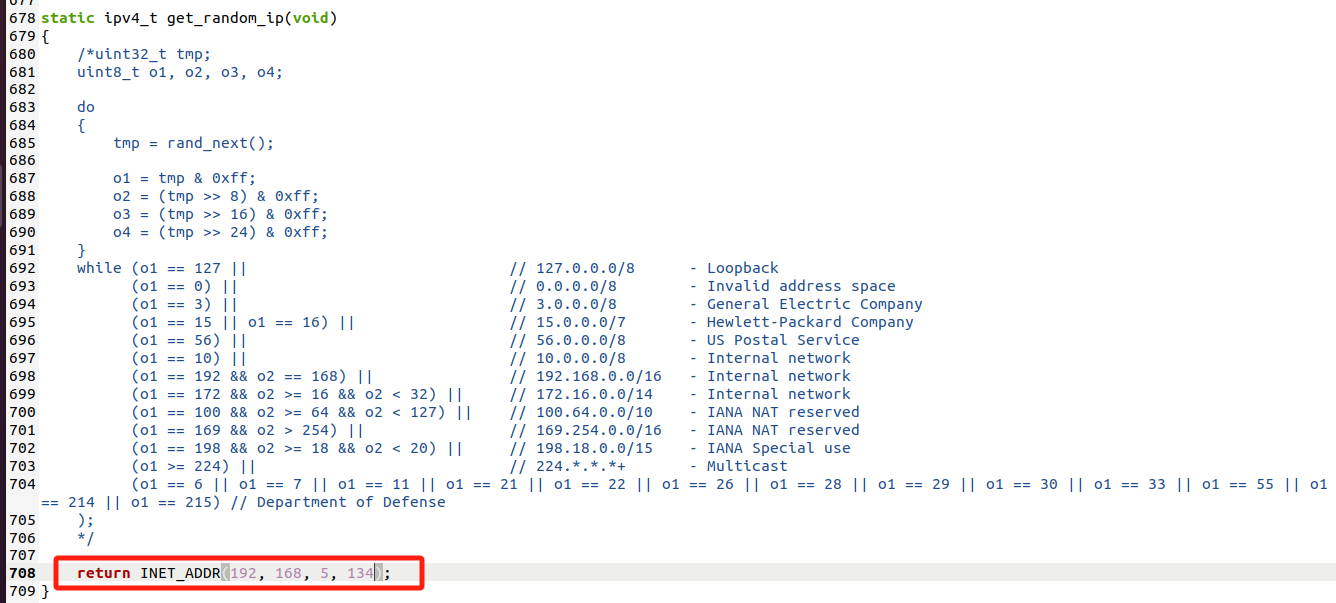
mirai就是根据这个返回字段“MIRAI: applet not found”来确认当前是否成功登录系统。假如当前没有成功进入对方系统，那么“enable”、“system”、“shell”、“sh”是进入不了交互模式的，此时输入“/bin/busybox MIRAI”也不会得到“MIRAI: applet not found”，只会得到telnet服务关于“登录失败”的提示语；而假如成功登录了系统，便会得到“MIRAI: applet not found”。

**分析完了，接下来是复现过程。**

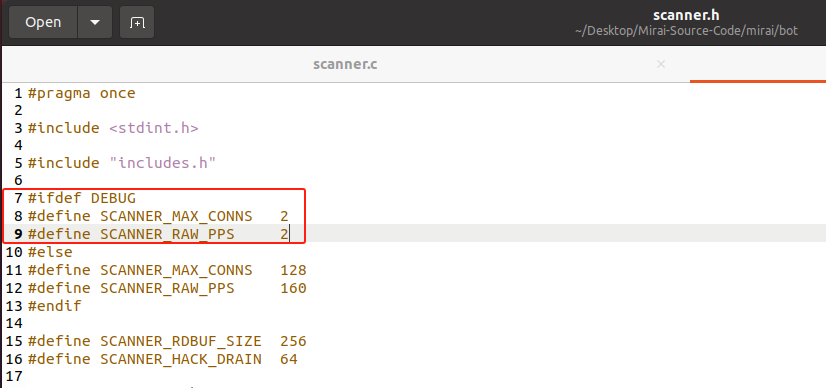
先将scanner.c中，协商次数进行修改。



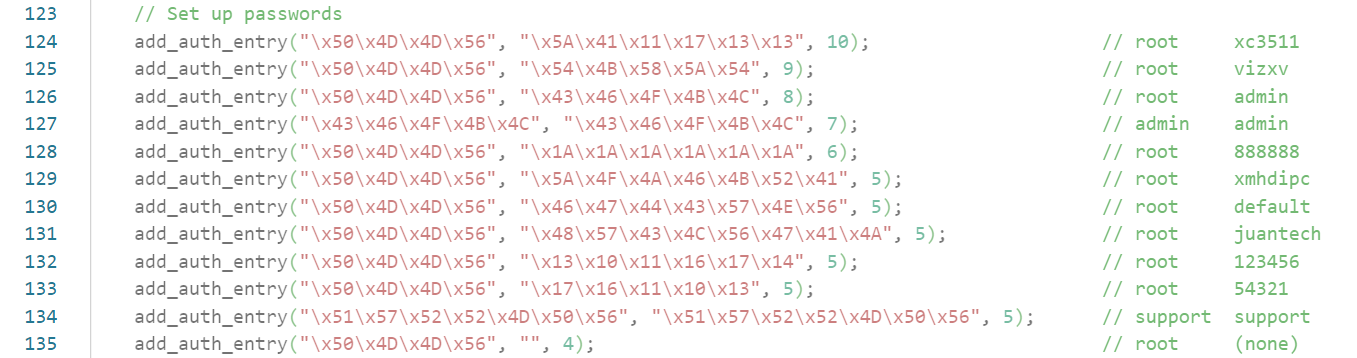
为了能扫描到指定ip，再去修改scanner.c里的函数get\_random\_ip，将返回的ip指向一个搭建好telnet服务器的待感染虚拟机，待感染虚拟机的ip为192.168.5.134。



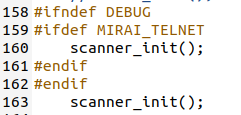
为了避免多线程扫描一台虚拟机，引起DoS，这里将线程数量降低成2个。



待感染虚拟机的账号密码为root、admin，确保他们出现在硬编码的账号、密码组中。

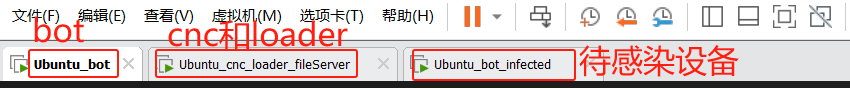


将扫描模块从条件宏中取出来。

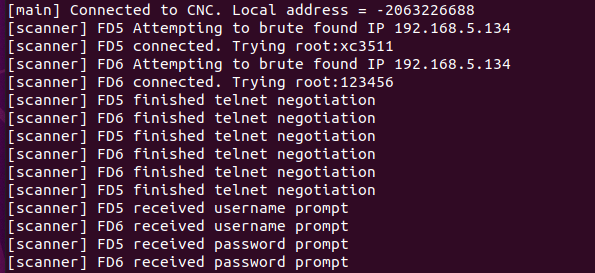


(.../mirai/bot/main.c)

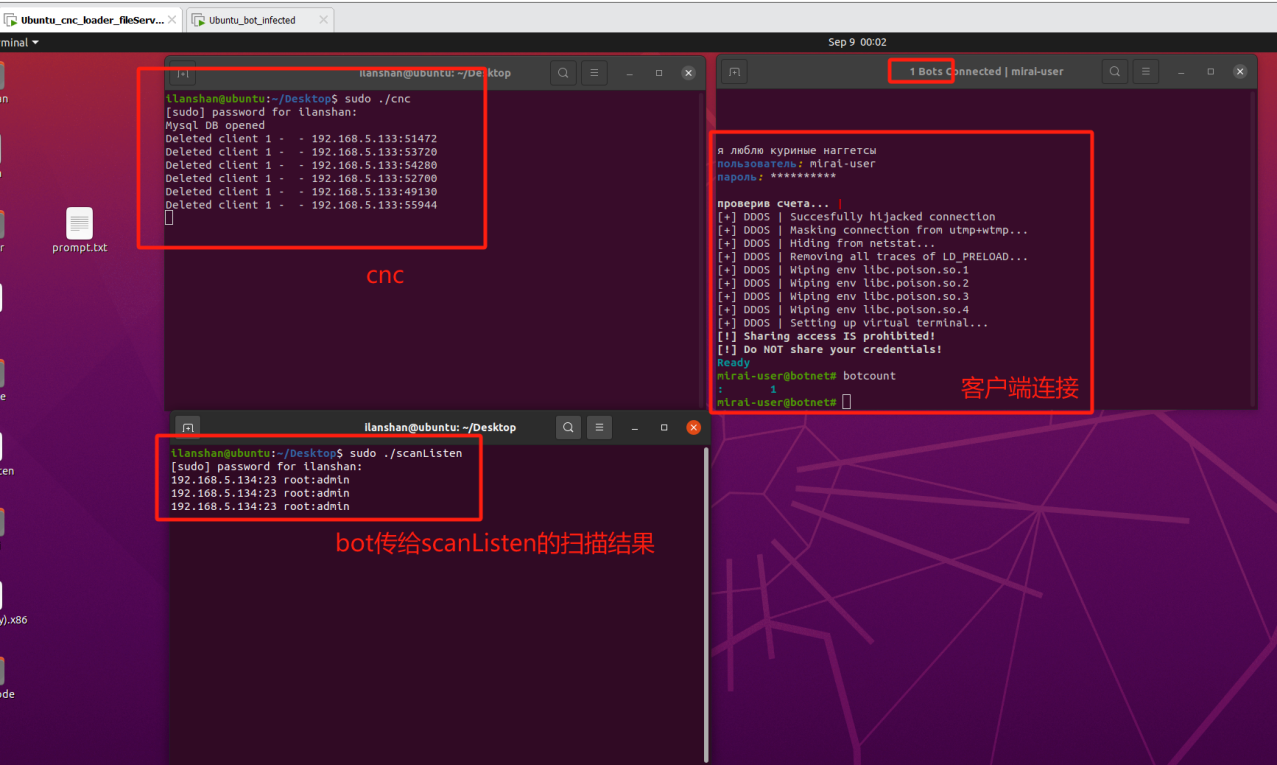
之后将bot重新编译并执行，并在域名report.mirai.com所对应的虚拟机上开启程序scanListen，接收bot返回的扫描结果。（Loader和CNC我放在同一个虚拟机上跑，Loader可以分为loader和scanListen，loader负责充当文件服务器，进行恶意程序的下发；scanListen用来接收bot暴力破解成功的账号密码及ip）



启动bot，将会连接上cnc.mirai.com，开始扫描ip地址，这里由于我指定扫描待感染设备192.168.5.134，所以FD5和FD6都是扫同一个ip。



bot会将成功扫描的结果发回给Loader主机的进程scanlisten。

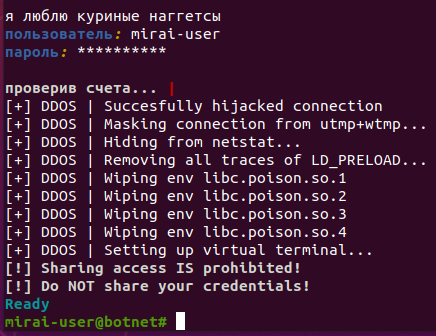


### 2.3 用户控制cnc，向bot发送DDoS攻击指令

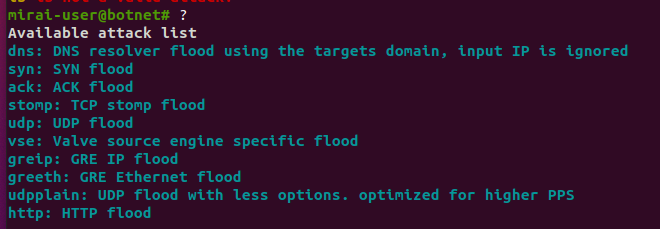
用户可以向cnc发送指令，cnc负责将指令转发给各个bot，以此完成用户向bot传达指令的任务。

用户可以通过telnet <域名 | cnc\_ip>去连接cnc主机，如果选择通过域名的方式，需要保证设置的dns服务器能够解析cnc的域名，这里由于我自己搭建了一个dns服务器，所以可以通过telnet cnc.mirai.com连接cnc主机。





输入?可以查看指令列表。

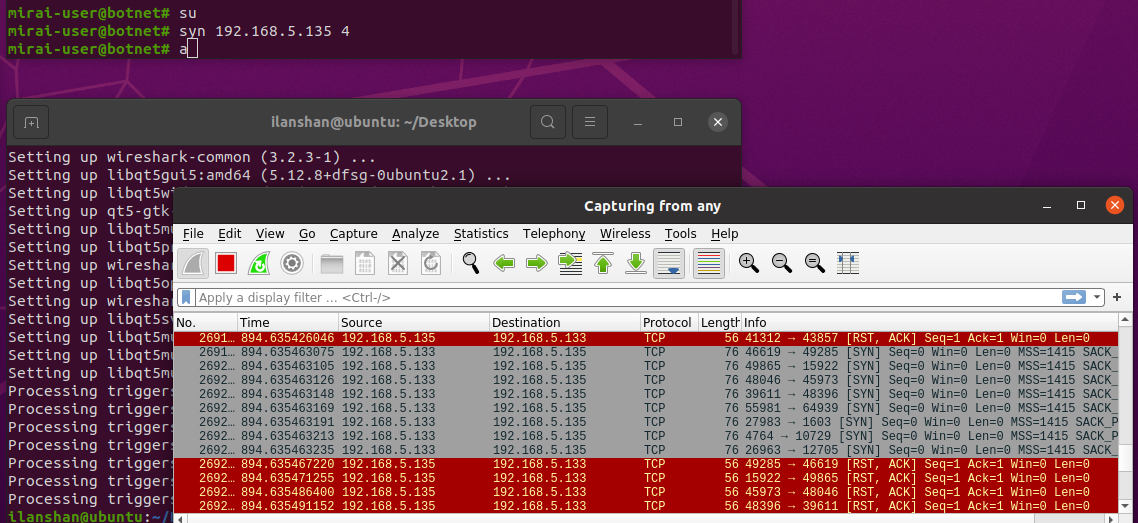


下发一个syn洪泛攻击，持续10s，目标主机为Ubuntu\_cnc\_loader\_fileServer，ip为192.168.5.135，在发送完指令后，可以发现连接cnc的bot数量变成了0，回到虚拟机bot，会发现虚拟机重启了。

这里需要修改一下攻击函数，比方说，刚才我们执行syn洪泛攻击死机了，执行syn洪泛攻击的函数是.../mirai/bot/attack\_tcp.c的attack\_tcp\_syn，将这里的break去掉，注释掉后，bot就不会断开连接，导致系统崩溃重启了。



然后重新编译bot，重新执行攻击指令，通过wireshark抓包观察攻击流量。



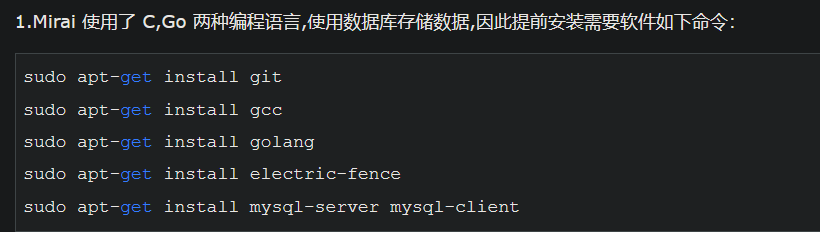
可以通过wireshark去分析流量特征，这里就不多言。如果相应执行其他种类的洪泛攻击，还需要去源码里面注释掉相应的break并重新编译。

## 3 搭环境

### 3.1 需要提前下载的软件

**网上的教程如下。**

在终端输入下列指令，准备环境。

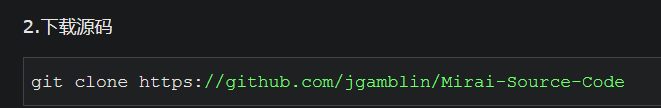


**我的操作如下。**

无图，照着打。

### 3.2 下载源码

**网上教程如下。**

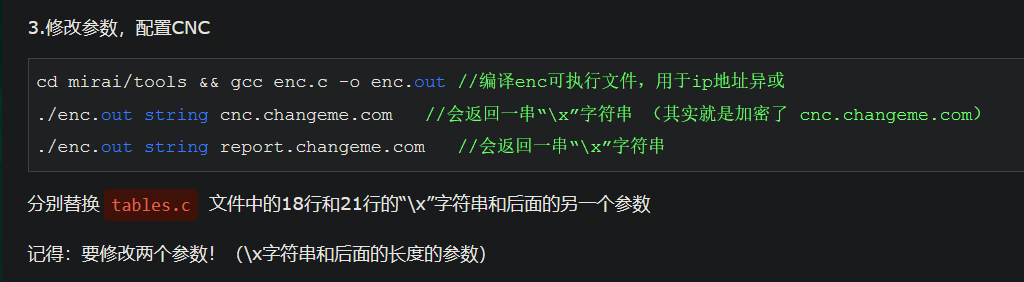
’

**我的操作如下。**

无图，照着打。如果这里下载不成功，记得开个代理，虚拟机如何开代理，可以去网上搜，或者在主机上下载，传到虚拟机里。

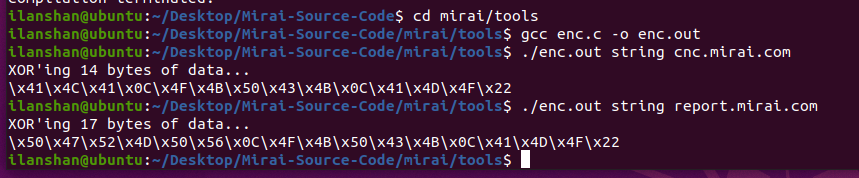
### 3.3 修改参数

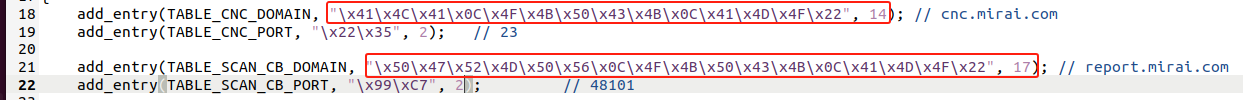
**网上的教程如下。**



**我的操作如下。**

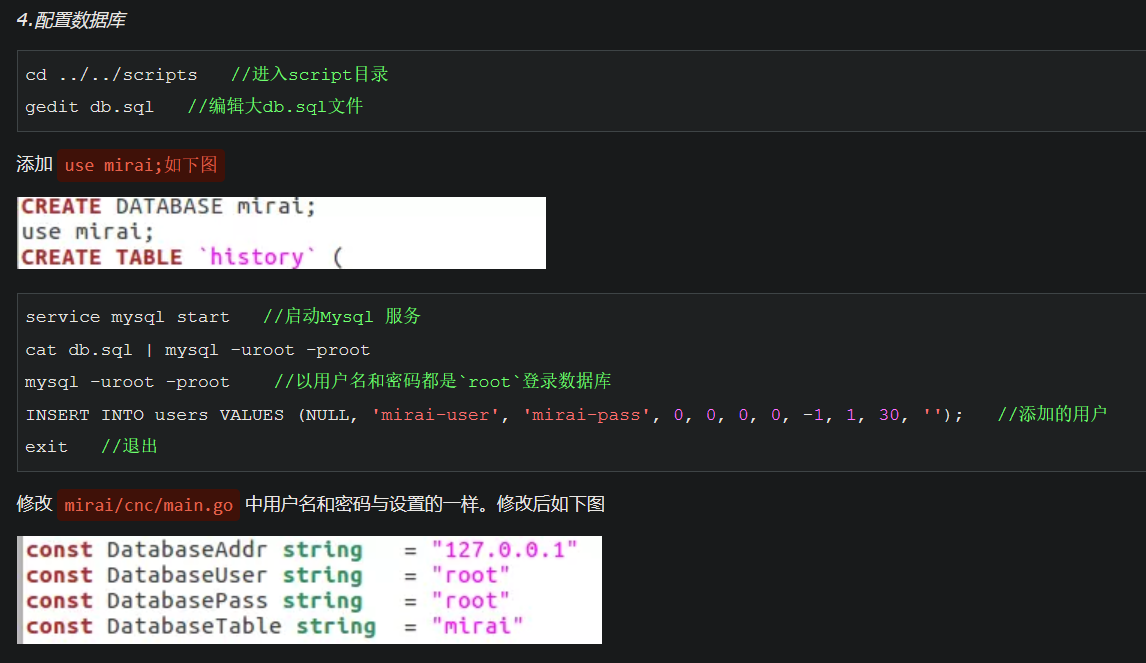
将生成的硬编码在.../bot/table.c中做替换。





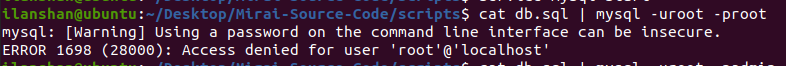
### 3.4 配置数据库

**网上的教程如下。**

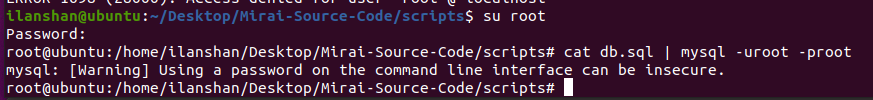


**我的操作如下。**

操作数据库时，被拒绝登录。



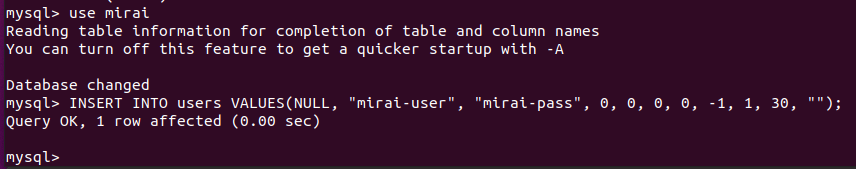
被拒绝访问了，或许是权限不够，切换成root再试试，这回成功执行了。



插入用户，提示没选择数据库。



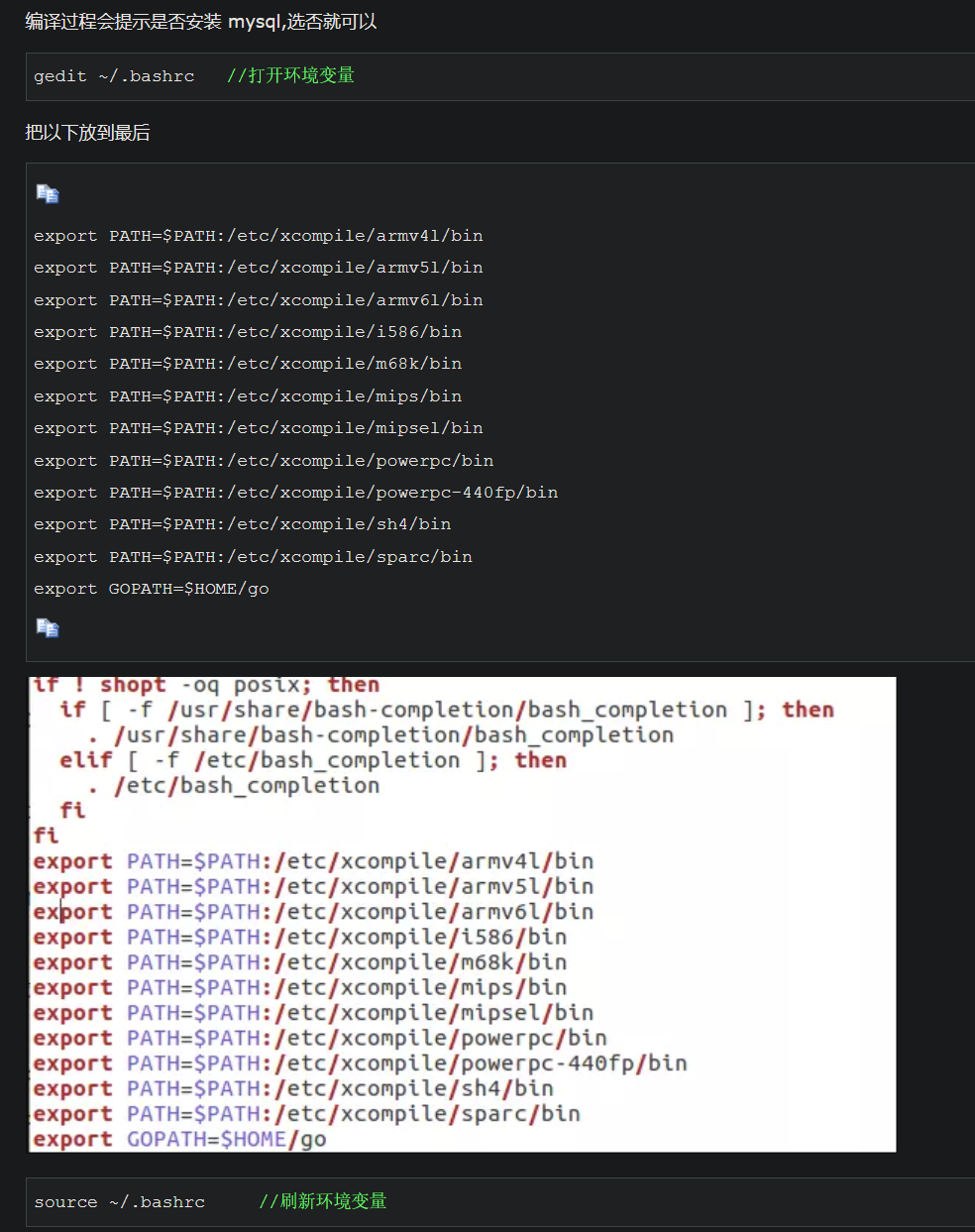
先选择mirai，再插入用户。



### 3.5 配置交叉编译环境

**网上的教程如下。**





**我的操作如下。**

1.照着网上的步骤走，下载所需要的包，添加环境变量，刷新环境变量。

2.交叉编译包如果从官网下不了，就去图中的百度云下。

3.添加环境变量时，root用户和普通用户的”~/.bashrc”是不一样的，可以在2个”~/.bashrc”的末尾，都添加上上述的路径。

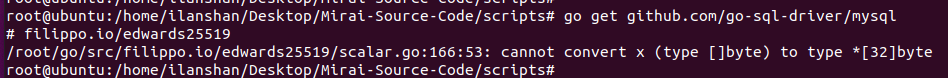
### 3.6 编译cnc和bot

**网上的教程如下。**



**我的操作如下。**

在执行go get ...的语句时，报错内容大致是某个类型无法转换成另一个类型。



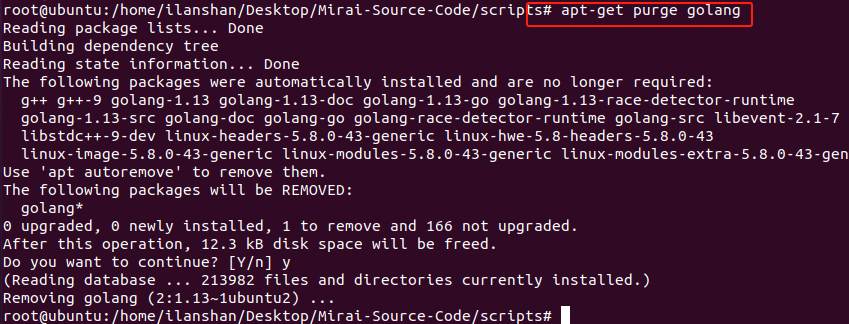
Google搜索，或者其它论坛，去搜这个问题。



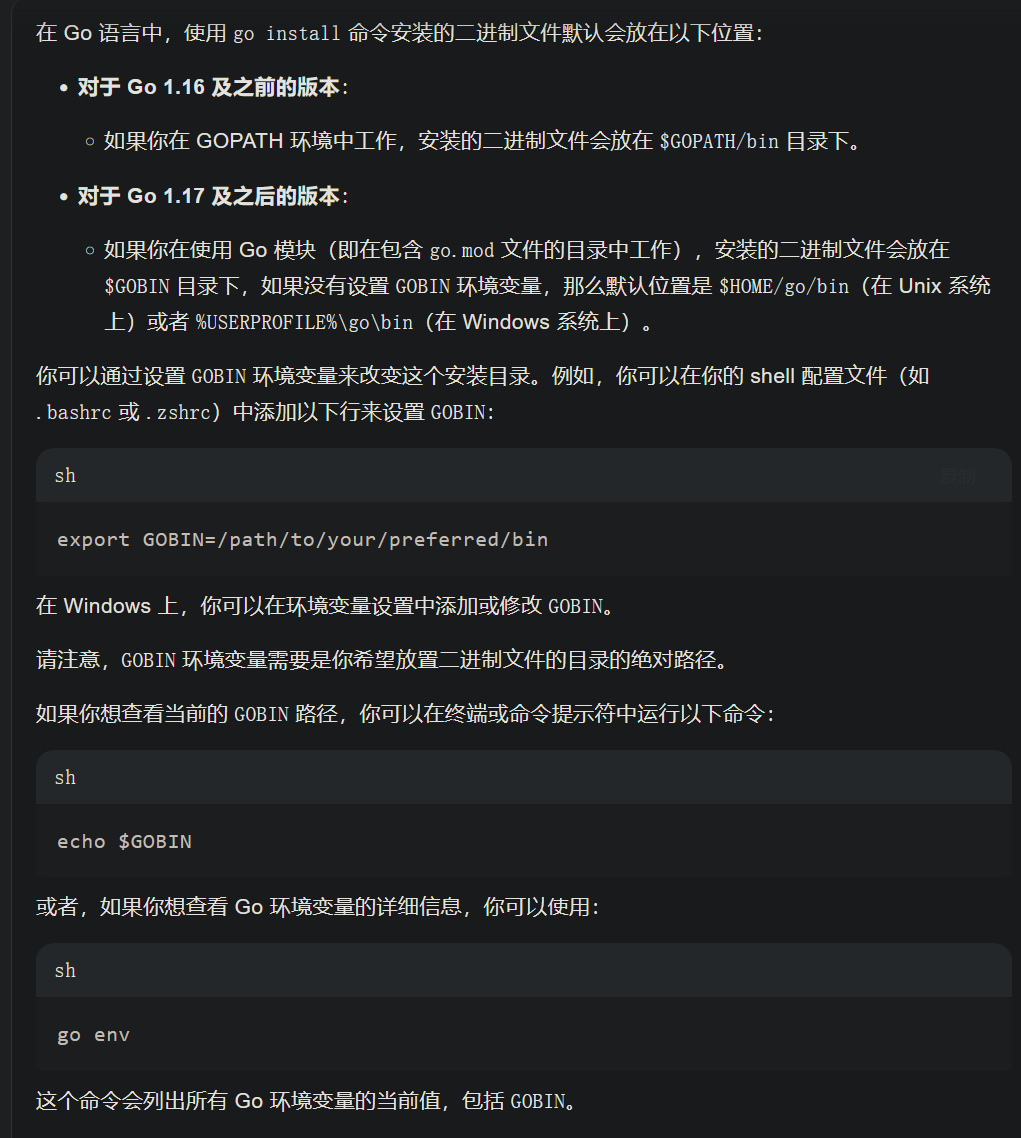
当前的go版本是1.13.8。



首先需要卸载当前的go，当前的go是通过apt-get下载安装的，如果不知道怎么卸载，继续问互联网。



下载最新版golang，Go的官网中，最新版是1.23.0，但这里我们不挑选最新版golang，原因如下。



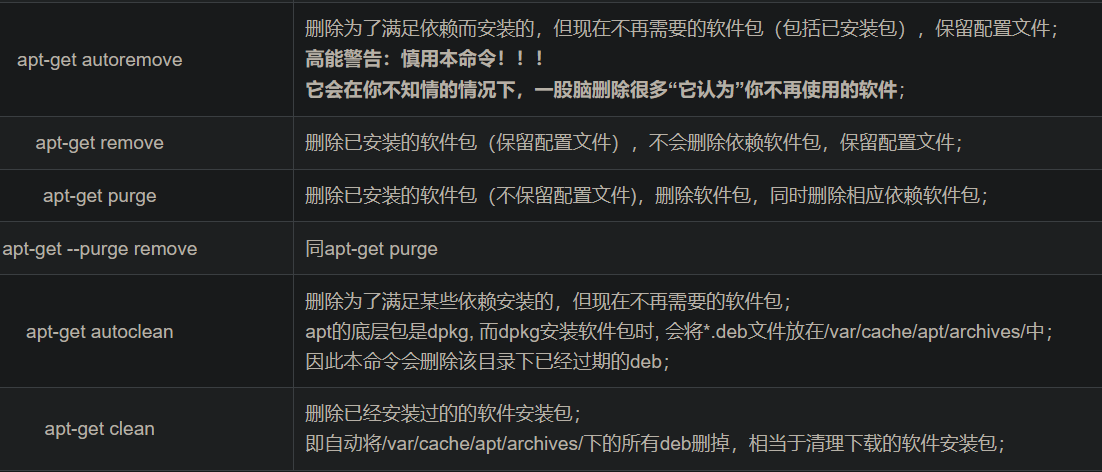
挑选对应架构的包下载即可，下载完后，照着下面的链接做即可。

<https://blog.csdn.net/qq_44026293/article/details/108699566>

有学弟问我，为什么装了最新的go，也修改了环境变量（~/.bashrc），但使用go version时仍然提示1.13版本。



不妨再看看删没删干净apt-get下载的go 1.13。



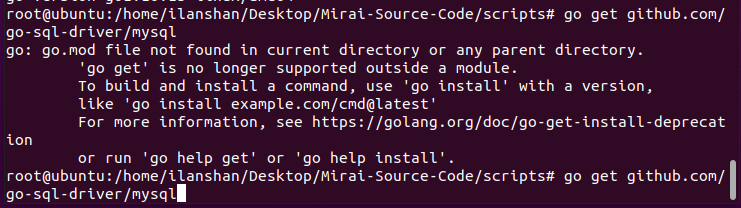
这里我用apt-get autoremove删除golang相关的依赖，然后刷新环境变量，再次使用go version，就能显示从官网安装的go版本了。



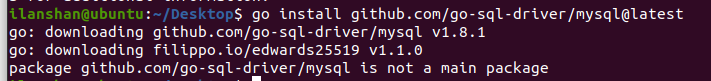
然而，这个bug需要将go的版本至少提升到go 1.20。



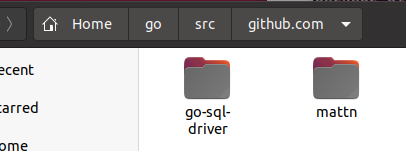
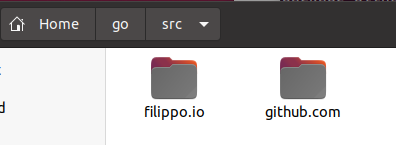
如果使用go 1.23.0，下载时会提示不再支持在模块外使用go get并推荐使用go install。



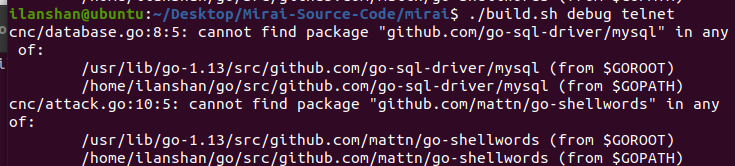
如果使用go get下载依赖包，默认的下载目录是$GOPATH/src，而go install不同，它会默认下载到$HOME/go/pkg/mod，然后编译安装这个模块，由于mysql不是一个可安装模块，这里显示的是xxx is not a main package。



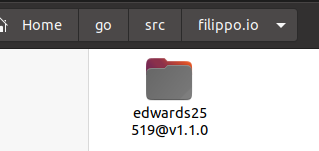
将这两个模块从pkg移到src。（或者直接从github下载，再导入这个文件夹）



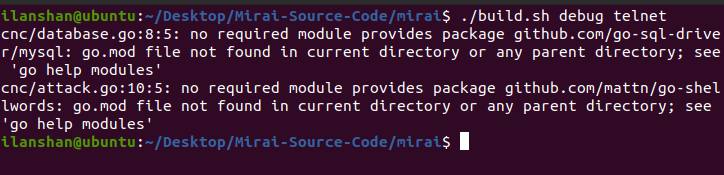
执行./build.sh debug telnet，提示找不到第三方包。



我回到目录.../go/src下看了一下，发现我下载的包都带着版本号，尝试把文件夹尾部的版本号去掉，重新执行。



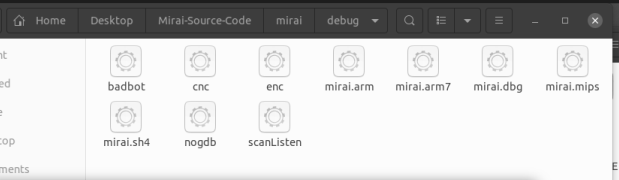
重新执行后，提示需要go.mod，估计是因为我用了最新版1.23.0的golang。



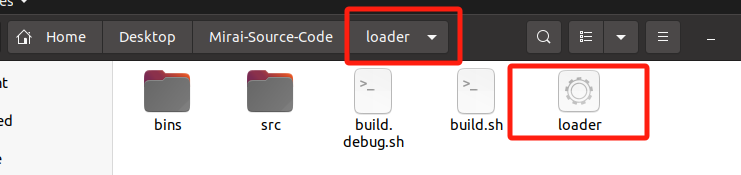
这里设置环境变量GO111MODULE=off，关闭模块模式。



重新执行./build.sh debug telnet，这回成功编译出了bot与cnc。



再切换到文件夹loader下编译loader。



至此，环境搭建完成。

### 3.7 总结

搭环境要有耐心，找到问题所在，然后解决，有问题尽量问互联网。

我第一次搭mirai的环境是在一、两年前，遇到的问题和现在不一样，这次搭环境遇到的问题主要来自Go的版本升级。希望学弟学妹不仅要会搭环境，还要学会如何解决遇到的问题，对点下药，效率会高很多。