

## 第 1 章

# 如何分析macOS软件

万事开头难。许多希望学习逆向工程的朋友通常在网上翻看了许多相关的博客和教程之后仍会觉得无从下手，第1章将会带你从头开始搭建一个最简单的分析环境，引导你自己动手写一个简单的CrackMe并破解它。

这一章不会介绍复杂的IDE，也不会使用各种“酷炫”的逆向工具，一切删繁就简，只使用命令行工具和简单的命令，来完成我们第一次Mac平台的逆向之旅，让你对逆向的过程有一个初步的认识。虽然目前在这个分析环境中只有几个命令行工具，但在后面的章节我们会不断地扩充。另外，Mac平台上的分析工具目前还不像Windows上那样种类繁多，所以我们还会自己开发一些实用的工具，添加到我们的分析环境中，使它变得更为充实。

## 1.1 分析环境搭建

首先，我们需要一个编译器来编译代码，目前在macOS系统上最流行的编译器自然是大名鼎鼎的Clang。

### 1.1.1 安装 Clang

如果你安装过Xcode，那说明你已经安装了Clang编译器，就可以先跳过本节。

Clang隶属于苹果公司的开源项目LLVM，是LLVM的一个前端，LLVM的官网为<http://llvm.org>，如图1-1所示。

你可以直接到LLVM官网下载编译好的Clang，但是这样下载Clang缺少一些必要的工具，使用起来很不方便，这些工具大部分跟Clang一样，包含在苹果的开发工具包中，这个工具包可以直接从App Store上下载。但是，还有更简便的方法。

首先打开一个终端，点击Launchpad→其他→终端，在终端中输入clang并回车，系统会自动检测到我们有没有安装Clang编译器，然后会提示我们是否下载并安装命令行开发者工具，如图1-2所示。



图1-1 LLVM官网



图1-2 安装命令行开发者工具提示

选择“安装”，就会出现安装协议，同意安装协议，过一会儿Clang编译器就会下载并安装到系统中了，一起安装的还有make等常用的命令行编译工具。我们可以执行clang -v查看Clang是否安装正确。

```
$ clang -v
Apple LLVM version 7.0.0 (clang-700.1.76)
Target: x86_64-apple-darwin14.5.0
Thread model: posix
```

如果能正确输出版本信息，则说明已经成功安装了Clang，接下来就可以使用它来编译程序了。

### 1.1.2 HT Editor

HT Editor是一个开源跨平台的十六进制编辑器，但它的功能可远远不止十六进制编辑器这么简单，它还有强大的反汇编/汇编功能，支持x86、x64、ARM、Power等多种处理器，并支持Windows平台上的PE文件格式、Linux上的ELF格式以及macOS的Mach-O文件格式。我们在这里要用它来破解CrackMe。

HT Editor的官网为<http://hte.sourceforge.net/>（如图1-3所示，打开该网站可能需要国外的代理）。



图1-3 HT Editor官网

HT Editor官网只提供了Windows版本的二进制文件，在Mac上我们需要自己动手编译来生成它。不过不要担心，这并不是什么难事，在macOS上编译大多数开源项目跟在UNIX系统是一样的，基本上只需要“./configure && make”就可以了。

首先，点击Downloads超链接，下载最新版本的源代码，如图1-4所示。



图1-4 HT Editor官网

在写作本书时，HT的最新版本是2.1.0。点击“[ht-2.1.0.tar.bz2](#)”超链接下载源代码并将其解压缩到一个合适的目录，我将其解压到用户目录下的Project目录中。然后打开一个终端，使用cd命令切换到源代码所在的目录，然后在终端中执行./configure并回车，会看到大量的“checking”：

```
$ cd Project/ht-2.1.0/
$ ./configure
checking build system type... x86_64-apple-darwin14.5.0
checking host system type... x86_64-apple-darwin14.5.0
checking target system type... x86_64-apple-darwin14.5.0
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... ./install-sh -c -d
checking for gawk... no
checking for mawk... no
checking for nawk... no
checking for awk... awk
checking whether make sets $(MAKE)... yes
checking whether make supports nested variables... yes
```

.....

```
config.status: creating minilzo/Makefile
config.status: creating output/Makefile
config.status: creating tools/Makefile
config.status: creating config.h
config.status: config.h is unchanged
config.status: executing depfiles commands
```

```
./configure successful.
```

```
=====
Configuration summary
=====
```

```
X11 textmode support available: no
enable profiling: no
make a release build: yes
using included minilzo: yes
```

如果一切顺利，会在终端输出“./configure successful.”，表示已经成功生成了编译需要的makefile文件。

---

提示 如果读者安装了X11，需要将命令改为./configure --disable-x11-text-mode禁用X11 textmode

的支持，否则可能会导致编译错误。

---

接下来输入make并回车，make程序会自动根据之前生成的makefile进行编译，并产生更多的编译输出。

```
$ make
/usr/bin/make all-recursive
Making all in tools
gcc -DHAVE_CONFIG_H -I. -I.. -DNOMACROS -O3 -fomit-frame-pointer -Wall -fsigned-char
-D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64 -MT bin2c.o -MD -MP -MF .deps/bin2c.Tpo -c -o bin2c.o
bin2c.c
bin2c.c:135:6: warning: variable 'iname' is used uninitialized whenever 'if' condition is false
[-Wsometimes-uninitialized]
    if (argc >= x + 1) {
        ^~~~~~
bin2c.c:140:11: note: uninitialized use occurs here
    in=fopen(iname, "rb");
        ^~~~~~
bin2c.c:135:2: note: remove the 'if' if its condition is always true
    if (argc >= x + 1) {
        ^~~~~~
bin2c.c:125:14: note: initialize the variable 'iname' to silence this warning
    char *iname, *outname, *outhname;
           ^
1 warning generated.
mv -f .deps/bin2c.Tpo .deps/bin2c.Po

...

mv -f .deps/htxexhead.Tpo .deps/htxexhead.Po
g++ -DHAVE_CONFIG_H -I. -I../analyser -I../asm -I../info -I../io/posix -I../io -I../output -I../eval -I.
-DNOMACROS -O3 -fomit-frame-pointer -Wall -fsigned-char -D_LARGEFILE_SOURCE
-D_FILE_OFFSET_BITS=64 -Woverloaded-virtual -Wnon-virtual-dtor -MT xexstruct.o -MD -MP
-MF .deps/xexstruct.Tpo -c -o xexstruct.o xexstruct.cc
mv -f .deps/xexstruct.Tpo .deps/xexstruct.Po
gcc -DHAVE_CONFIG_H -I. -I../analyser -I../asm -I../info -I../io/posix -I../io -I../output -I../eval -I. -DNOMACROS
-O3 -fomit-frame-pointer -Wall -fsigned-char -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64 -MT
cp-demangle.o -MD -MP -MF .deps/cp-demangle.Tpo -c -o cp-demangle.o cp-demangle.c
mv -f .deps/cp-demangle.Tpo .deps/cp-demangle.Po
g++ -DHAVE_CONFIG_H -I. -I../analyser -I../asm -I../info -I../io/posix -I../io -I../output -I../eval -I.
-DNOMACROS -O3 -fomit-frame-pointer -Wall -fsigned-char -D_LARGEFILE_SOURCE
-D_FILE_OFFSET_BITS=64 -Woverloaded-virtual -Wnon-virtual-dtor -MT htxeximg.o -MD -MP
-MF .deps/htxeximg.Tpo -c -o htxeximg.o htxeximg.cc
mv -f .deps/htxeximg.Tpo .deps/htxeximg.Po
g++ -DNOMACROS -O3 -fomit-frame-pointer -Wall -fsigned-char -D_LARGEFILE_SOURCE
-D_FILE_OFFSET_BITS=64 -Woverloaded-virtual -Wnon-virtual-dtor -o ht atom.o except.o data.o str.o strtools.o
endianess.o htdoc.o block.o cstream.o formats.o htanaly.o htapp.o htcfg.o htclipboard.o htcoff.o htcoffhd.o htctrl.o
htdebug.o htdialog.o htelf.o htelfhd.o htelfimg.o htelfphs.o htelfshs.o htelfsym.o htelfrel.o htinfo.o htformat.o htex.o
```

```
hthist.o htidle.o htiobox.o htie.o htleent.o htlehead.o htleimg.o htleobj.o htleepage.o htmnu.o htmz.o htmzhead.o
htmzim.o htmzrel.o htne.o htneent.o htnehead.o htneoms.o htneobj.o htnewexe.o htobj.o htpe.o htpedimp.o
htpeexp.o htpehead.o htpeimg.o htpeimp.o htperes.o htpereloc.o htreg.o htsearch.o httag.o httree.o main.o store.o
stream.o tools.o vxd.o vxdserv.o cplus-dem.o regex.o syntax.o textfile.o textedit.o classread.o classview.o httex.o
hteval.o relfile.o htprocess.o mfile.o elfstruc.o pestruct.o coff_s.o mzstruct.o defreg.o htdisasm.o htcoffimg.o nestruct.o
htneimg.o htneimp.o cmds.o snprintf.o htpeil.o ilstruct.o log.o classimg.o vfs.o vfsview.o htlevxd.o lestruct.o htmach.o
htmachohd.o machostruc.o htmachimg.o fltstruc.o htflt.o htflthd.o htflimg.o xbestruct.o htxbehead.o htxbe.o
htxbeimg.o htxbeimp.o pefstruc.o htpef.o htpefhd.o htpefimg.o htpefimp.o htxex.o htxexhead.o xexstruct.o
cp-demangle.o htxeximg.o analyser/libanalyser.a asm/libasm.a info/libinfo.a io/posix/libhtio.a output/liboutput.a
io/libcomio.a eval/libhteval.a -lncurses minilzo/liblzo.a
```

Clang会以彩色输出显示编译中的警告和错误，一般警告为鲜红色，错误为暗红色。编译完成后如果没有错误，就可以在源代码根目录中看到编译好的HT。

### 1.1.3 Homebrew

很多时候，需要在系统中安装一些命令行工具与脚本，比如，安装wget或curl作为命令行下的下载工具，安装Git版本控制软件来管理工程的源代码。比较传统的安装方式是到这些软件的官网上下载编译好的程序，或者下载源代码进行编译，然后将程序放到一个目录下，将路径添加到PATH环境变量下，以后在终端中直接输入命令就可以使用它们。如果所有这些工具都使用这种方式安装，不但不方便进行管理，而且需要花费太多时间去搜索与安装它们。在主流的UNIX系统上，一般都有对这类软件进行统一管理的工具，如Ubuntu系统的apt-get，安装wget只需要在Ubuntu的终端中执行sudo apt-get install wget，就会在系统中自动安装wget。

苹果系统并没有提供这样的管理工具，但幸运的是，已经有第三方开发人员开发了这样的工具，并免费供用户使用。主流的有Homebrew与Macports，这里以Homebrew的使用为例。安装Homebrew只需要在终端中执行下面的代码即可：

```
$ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

执行后会产生如下输出：

```
=> This script will install:
/usr/local/bin/brew
/usr/local/share/doc/homebrew
/usr/local/share/man/man1/brew.1
/usr/local/share/zsh/site-functions/_brew
/usr/local/etc/bash_completion.d/brew
/usr/local/Homebrew
=> The following new directories will be created:
/usr/local/Cellar
/usr/local/Homebrew
/usr/local/Frameworks
/usr/local/bin
/usr/local/etc
/usr/local/include
/usr/local/lib
/usr/local/opt
/usr/local/sbin
/usr/local/share
/usr/local/share/zsh
/usr/local/share/zsh/site-functions
/usr/local/var

Press RETURN to continue or any other key to abort
=> /usr/bin/sudo /bin/mkdir -p /usr/local/Cellar /usr/local/Homebrew /usr/local/Frameworks
/usr/local/bin /usr/local/etc /usr/local/include /usr/local/lib /usr/local/opt /usr/local/sbin /usr/local/share
/usr/local/share/zsh /usr/local/share/zsh/site-functions /usr/local/var

WARNING: Improper use of the sudo command could lead to data loss
or the deletion of important system files. Please double-check your
typing when using sudo. Type "man sudo" for more information.

To proceed, enter your password, or type Ctrl-C to abort.

Password:
=> /usr/bin/sudo /bin/chmod g+rxw /usr/local/Cellar /usr/local/Homebrew /usr/local/Frameworks
/usr/local/bin /usr/local/etc /usr/local/include /usr/local/lib /usr/local/opt /usr/local/sbin /usr/local/share
/usr/local/share/zsh /usr/local/share/zsh/site-functions /usr/local/var
=> /usr/bin/sudo /bin/chmod 755 /usr/local/share/zsh /usr/local/share/zsh/site-functions
=> /usr/bin/sudo /usr/sbin/chown mbp /usr/local/Cellar /usr/local/Homebrew /usr/local/Frameworks
/usr/local/bin /usr/local/etc /usr/local/include /usr/local/lib /usr/local/opt /usr/local/sbin /usr/local/share
/usr/local/share/zsh /usr/local/share/zsh/site-functions /usr/local/var
=> /usr/bin/sudo /usr/bin/chgrp admin /usr/local/Cellar /usr/local/Homebrew /usr/local/Frameworks
/usr/local/bin /usr/local/etc /usr/local/include /usr/local/lib /usr/local/opt /usr/local/sbin /usr/local/share
/usr/local/share/zsh /usr/local/share/zsh/site-functions /usr/local/var
=> /usr/bin/sudo /bin/mkdir -p /Users/mbp/Library/Caches/Homebrew
=> /usr/bin/sudo /bin/chmod g+rxw /Users/mbp/Library/Caches/Homebrew
=> /usr/bin/sudo /usr/sbin/chown mbp /Users/mbp/Library/Caches/Homebrew
=> Searching online for the Command Line Tools
=> /usr/bin/sudo /usr/bin/touch /tmp/.com.apple.dt.CommandLineTools.installondemand.in-progress
.....
```



在安装过程中会创建系统的目录与软链接，如果系统没有安装Command Line Tools，则会自动下载安装。安装完成后，执行brew doctor命令可以查看Homebrew的环境是否正常。通常在第一次安装brew之后，还需要安装苹果的Command Line Tools。如果事先安装过Xcode，Command Line Tools会在Xcode第一次启动时提示安装。

安装好Homebrew后，执行以下命令就可以安装指定的软件包。

```
$ brew install 软件包名
```

更新Homebrew所有的软件可以执行以下命令。

```
$ brew update
$ brew upgrade
```

卸载指定的软件包可以执行：

```
$ brew remove 软件包名
```

上一节介绍的HT Editor在Homebrew中也有移植版本，只需要执行以下命令即可自动安装。

```
$ brew install ht
==> Installing dependencies for ht: lzo
==> Installing ht dependency: lzo
==> Downloading https://homebrew.bintray.com/bottles/lzo-2.09.el_capitan.bottle.tar.gz
#####
##### 100.0%
==> Pouring lzo-2.09.el_capitan.bottle.tar.gz
/usr/local/Cellar/lzo/2.09: 29 files, 565.0K
==> Installing ht
==> Downloading https://homebrew.bintray.com/bottles/ht-2.1.0.el_capitan.bottle.tar.gz
#####
##### 100.0%
==> Pouring ht-2.1.0.el_capitan.bottle.tar.gz
/usr/local/Cellar/ht/2.1.0: 8 files, 2.0M
```

Homebrew会依次下载安装HT Editor的依赖库，然后安装HT Editor并配置好它的路径，整个安装过程不需要用户手动干预，安装完成后，在终端下执行ht就可以打开HT Editor了。

此外，还有一个基于Homebrew的扩展工具Homebrew-Cask，使用它可以直接下载App Store中的GUI程序，安装Homebrew-Cask需要执行如下命令：

```
$ brew tap phinze/cask
$ brew install brew-cask
```

完成后就可以安装其他软件了，如安装腾讯的QQ聊天工具就可以执行如下命令：

```
$ brew cask install qq
```

如果你觉得在命令行下管理软件不够直观，可以尝试基于Homebrew移植的GUI版本CakeBrew，只需要到它的官网<https://www.cakebrew.com>下载安装即可。CakeBrew运行效果如图1-5所示。



图1-5 CakeBrew运行效果

## 1.2 第一个 macOS 程序

现在我们已经装好了编译环境，并使用这个编译环境编译了一个开源项目，接下来就要用这

个环境编写CrackMe了。

首先，我们需要一个编辑器编写代码。这里为了演示方便，选择的是系统自带的Vim编辑器，读者也可以自行选择其他方便的编辑器（后面会介绍更多实用的编辑器，本书附录的工具一览表中也会列出它们）。

我并不希望读者从一开始就陷入复杂的IDE的细节中，而是希望大家将重点放到代码的编写和逆向分析过程上。关于Xcode等IDE的使用，会在第3章讲解软件开发时进行详细的介绍。

首先，在Project文件夹下创建一个CrackMe01文件夹，里面存放编写的源代码以及编译结果：

```
$ cd Project
$ mkdir CrackMe01 && cd CrackMe01/
```

创建CrackMe的源代码文件cm01.c：

```
$vim cm01.c
```

这时可以看到出现了Vim的命令行界面，并且左下角有“cm01.c” [New File]”的提示，此时在键盘上按“a”键进入编辑模式。“a”键表示在当前光标位置之后插入字符，之后就可以像正常编辑器一样输入代码了，但是需要注意的是，数字不能用小键盘输入。

```
#include <stdio.h>

int main()
{
    int secret = 0;
    printf("Please enter the secret num:");
    scanf("%d",&secret);
    if (secret != 123)
    {
        printf("Incorrect secret num.\n");
        return 0;
    }

    printf("Hello world!\n");
}
```

```
    return 0;
}
```

代码很简单，先要求用户输入一个数字，然后判断这个数字是否是“123”，如果不是则输出“Incorrect secret num.”，如果是则输出“Hello world!”。输入完成后按“esc”键退出编辑模式，输入:wq命令保存并退出Vim，然后使用Clang编译。

```
$clang cm01.c -o cm01
```

如果没有编译错误，就会生成cm01可执行文件；如果有错误，可以重新执行vim cm01.c命令，回到Vim中编辑代码的错误之处。然后测试一下CrackMe运行是否正常。

```
$ ./cm01
Please enter the secret num:456
Incorrect secret num.
$ ./cm01
Please enter the secret num:123
Hello world!
$
```

当输入的数字不是“123”的时候，程序输出“Incorrect secret num.”，如果是“123”则输出“Hello world!”，说明程序运行正常。

接下来就要破解这个简单的CrackMe了，让它在我们输入任意值的时候都会输出“Hello world!”。

## 1.3 使用 HT Editor 进行破解

终于到破解环节了！如果这是你初次接触Mac上的破解，那么下面就将是第一次亲手破解的程序。

这一节会用到少量的汇编指令和Mach-O文件格式的相关知识，后面的章节会详细讨论

Mach-O文件格式和x86\_64汇编指令，这里我们只需要知道几条关键汇编指令的含义即可，其余的都让HT Editor帮我们进行处理。

- jz指令：跳转指令，可以理解成如果前面比较指令的比较结果相同则跳转到指定的地址，如果不相等就不跳转，继续执行它下面的指令；
- jnz指令：与jz指令正好相反，不相等则跳转；
- jmp指令：不管任何情况都会进行跳转；
- call指令：调用过程指令，一般对应高级语言中的函数调用。

这些跳转指令的作用是什么呢？我们用高级语言所写的代码，最终都会被编译器翻译成机器码，其中的判断语句一般翻译成跳转指令，所以我们可以通过修改跳转指令达到修改软件的执行流程的目的，最终让目标程序跳过检查，执行我们所期望的代码。

有了理论基础，下面就该开始动手实践了。首先用HT Editor打开要破解的程序，也就是我们自己写的cm01。首先打开一个终端，并通过cd命令切换到cm01所在的文件夹，然后在终端中运行ht命令，会启动HT Editor的命令行界面，如图1-6所示。

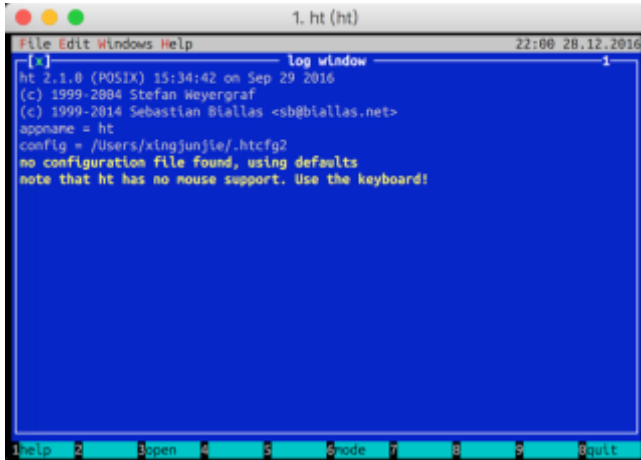


图1-6 HT Editor主界面

HT Editor在POSIX兼容系统上有些按键会有问题，需要用esc代替ctrl键，所以主菜单就变成了esc+红色字母，而下面的1到0快捷键分别对应的是fn+F1到fn+F10。首先按fn+F3 ( open )，会出现选择文件的界面，如图1-7所示。

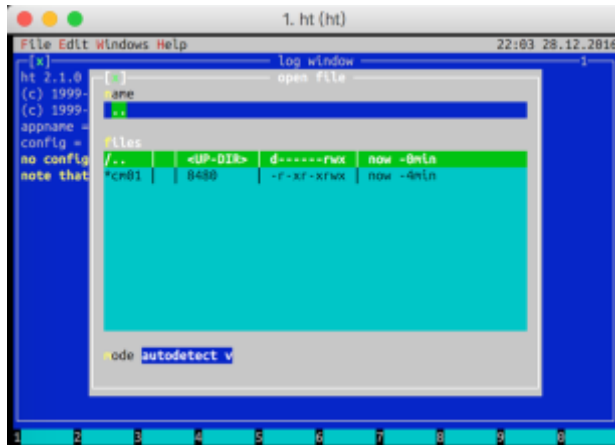


图1-7 选择文件

然后按tab键将光标移到下面的列表框，找到cm01并回车，就会出现十六进制编辑界面，如



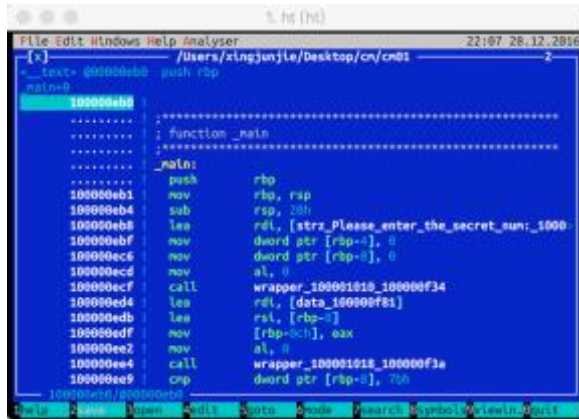


图1-10 HT反汇编界面

有了反汇编代码，我们就需要阅读反汇编并找到判断数字是否正确的关键跳，并对其进行修改。HT为我们识别出了main函数，我们可以顺着main函数理清逻辑，这个CrackMe很简单，我们这么做也不是特别费力，但是如果代码比较复杂，这种方法就不适用了。

还有更简单的方法，当我们输入一个错误的数字的时候，程序会提示一句 “Incorrect secret num”，我们可以用这句提示作为突破口。HT提供了一个搜索 ( search ) 功能，我们可以使用这个功能来查找这句错误提示。

首先，按快捷键fn+F7，出现搜索对话框，mode选择 “display: regex”，表示使用正则表达式搜索HT界面中显示的字符，e为搜索的正则表达式，不过我们目前还用不到复杂的正则表达式，只需要输入 “incorrect” 就可以了。选中 “case insensitive ( 不区分大小写 )”，如图1-11所示。



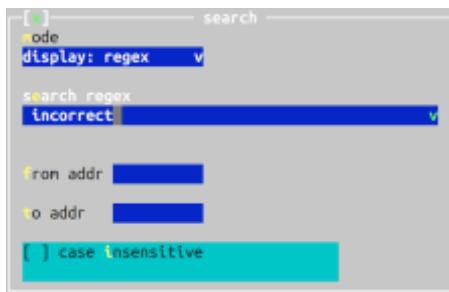


图1-11 HT搜索对话框

然后按回车，光标会定位到下面这行代码：

```
100000ee9 ! lea rdi, [strz_Incorrect_secret_num.__100000f78]
```

这句汇编代码的意思是将100000f78这个地址放入rdi寄存器，100000f78中的内容是一个以0结尾的字符串（strz），内容为“Incorrect secret num.”，HT自动将内容中的空格换成了下划线。

按shift+fn+F7可以继续搜索，但是在Mac上这个快捷键似乎无效。

我们到100000f78这个地址这里就可以看到“Incorrect secret num.”这个字符串。找到了这个字符串，那它上面的代码就应该是判断的跳转，往上一看，它上面就是如下代码：

```
100000ee3 ! jz loc_100000f06
```

到100000f06处进行查看，会发现如下代码：

```
100000f06 !
..... ! loc_100000f06: ;xref j100000ee3
..... ! lea rdi, [strz_Hello_world__100000f8f]
100000f0d ! mov al, 0
100000f0f ! call wrapper_100001010_100000f28
```

这里将“Hello world”字符串的地址传给了一个函数，不难猜测，这个函数应该就是printf，到这里就应该是成功了。地址100000ee3处的跳转就是跳向成功的关键跳，也就是我们要修改的地方。

回到100000ee3处,按快捷键ctrl+a修改这里的汇编代码,修改成jnz loc\_100000f06,将原来的相等则跳转改成不相等则跳转,如图1-12所示。

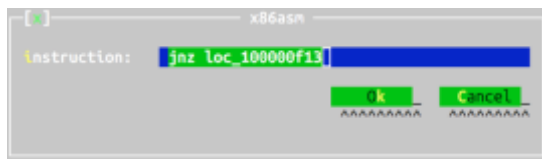


图1-12 HT修改汇编代码对话框

回车后,HT会列出所写的汇编代码可以翻译成的机器码,因为x86和x64一条汇编语句可能对应多种机器码,这里选择和原来机器码长度相同的机器码最合适,既不需要填充nop指令,也不会覆盖后面的机器码。HT默认为我们选中了最合适的机器码,直接回车即可,如图1-13所示。

可以看到,实际上只修改了一个字节的机器码(变红的85),但此时的修改还没有保存到磁盘文件,按fn+F2进行保存。保存完后,红字就消失了,我们在来试试下面这个CrackMe程序。

```
$ ./cm01
Please enter the secret num:456
Hello world!
$ ./cm01
Please enter the secret num:123
Incorrect secret num.
```

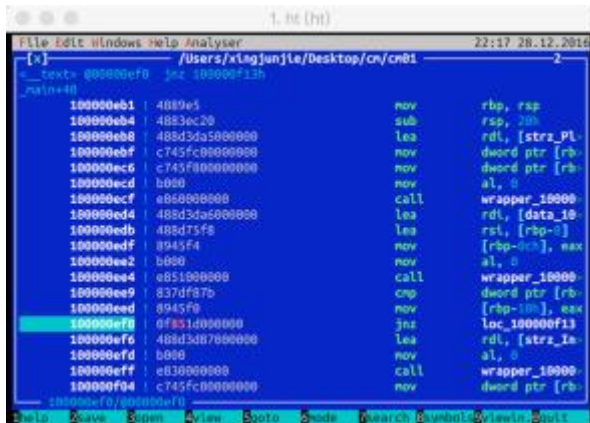


图1-13 HT修改结果

如果我们输入一个错误的数字，会输出“Hello world!”，输入正确的反倒提示失败了，这说明我们的修改成功了！

不过还是有点不完美，我们希望输入正确的数字也要提示成功。虽然在实际使用中乱猜正好输入了正确的数字可能性太低了，但是我们要在技术上追求完美。回到HT，找到跳转指令，按ctrl+a快捷键，这次我们把jnz改成jmp，让它无论数字正确与否都直接跳到打印“Hello world!”处，回车保存，再次测试如下。

```
$ ./cm01
Please enter the secret num:74551122
Hello world!
$ ./cm01
Please enter the secret num:123
Hello world!
$ ./cm01
Please enter the secret num:6697
Hello world!
```

哈！不管输入什么都会提示成功了！第一次的破解之旅到此完美结束！

## 1.4 本章小结

本章介绍了最基本的分析环境搭建和一个简单程序的破解。通过本章的学习，相信你已经对macOS系统上的软件逆向分析技术有基本的认识了。