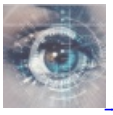
使用objcopy选择性修改删除符号表以及其他相关elf段以及gdb debug file 任意读漏洞以及xctf mamadebug writeup

原创

fjh1997 于 2021-05-31 18:27:51 发布 278 收藏

分类专栏： 安全 CTF

本文链接：https://blog.csdn.net/fjh1997/article/details/117420338



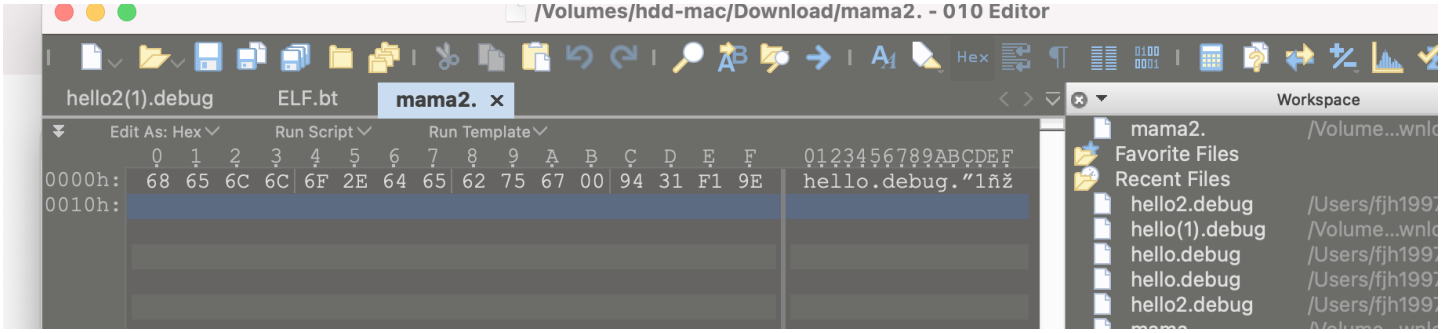
由于一个elf文件里面的调试信息和符号信息占用了很大的空间，而这些信息只是调试的时候才用到，平时运行的时候用不到。所以我们可以把这部分信息单独dump出来作为一个独立的文件。开发者在需要调试的时候可以下载这个文件来恢复elf的调试信息和符号表。

```
objcopy --only-keep-debug foo foo.debug
#将elf文件的调试信息单独dump出来作为一个独立的elf文件foo.debug
```

但是怎么知道这个elf文件的debug文件是什么呢？这就要参考：https://sourceware.org/gdb/current/onlinedocs/gdb/Separate-Debug-Files.html

根据这篇文章，elf文件对应的debug文件在.gnu_debuglink段里面。分别指定了文件名和文件的crc.
我们可以dump出.gnu_debuglink段来查看:

```
objcopy –dump-section .gnu_debuglink=mama2   hello
#dump某个elf段到某个文件中，比如上述例子是把.gnu_debuglink段dunmp到mama2这个文件中
```



可以看到debug文件的文件名是hello.debug，其crc32是94 31 F1 9E，由于是小端序的，所以crc值是0x9EF13194。
如果想更换debug文件，就要删除.gnu_debuglink段然后重新添加：

```
objcopy --remove-section=.gnu_debuglink  hello #删除某个elf段，比如.gnu_debuglink段
```

```
objcopy --add-gnu-debuglink=foo.debug hello
#添加了foo.debug作为调试文件。
```

接下来我们来看xctf的一道题：

```
import functools
import sys
import tempfile
import shutil
import os
import subprocess

print = functools.partial(print, flush=True)

def main():
    print('What do you think should be present in a debug info file ?')
    print('Give me your answer, I will debug it for you :)')
    print('File > ')

    content = sys.stdin.buffer.read(0x1024)
    print(len(content))
    with tempfile.TemporaryDirectory() as tempdir:
        bin_path = os.path.join(tempdir, 'hello')
        shutil.copyfile('/home/ctf/hello', bin_path)
        with open(os.path.join(tempdir, 'hello.debug'), 'wb') as f:
            f.write(content)
        os.chdir(tempdir)
        os.chmod(bin_path, 0o755)
        subprocess.run(['gdb', 'hello','-ex','set confirm off', '-ex', 'b main', '-ex', 'r', '-ex', 'p a', '-ex'
, 'c','-ex','q'])


if __name__ == '__main__':
    main()
```

这道题首先限制了debug文件的大小要小于4k。其次，这道题会将你输入的文件作为debug文件进行调试。同时也会校验你debug文件与hello文件里面指定的debug文件的crc是否相等。

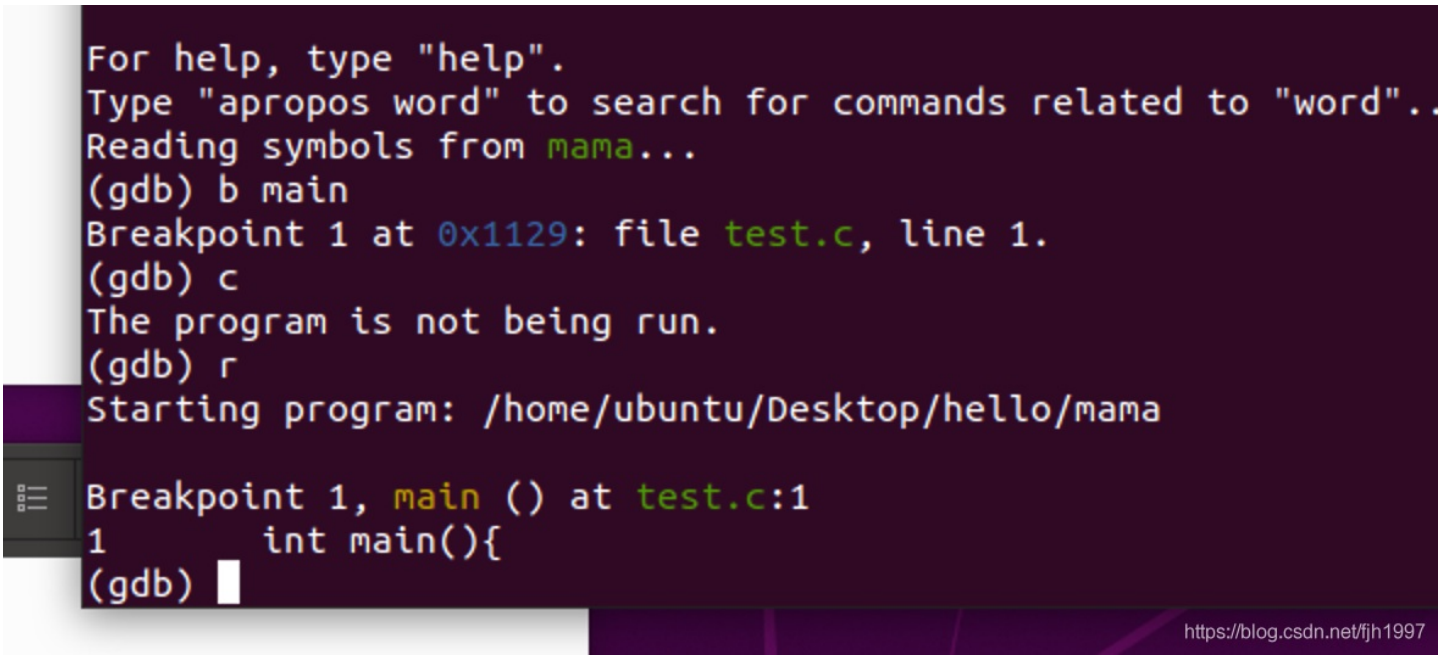
这道题的正确解法是这样的:
首先我们创建一个c语言文件,不引入任何头文件，为了确保足够小:

```
int main(){
 return 0;
}
```

如果我们在gcc编译的时候加上`-g`参数，

```
gcc -g test.c -o mama
```

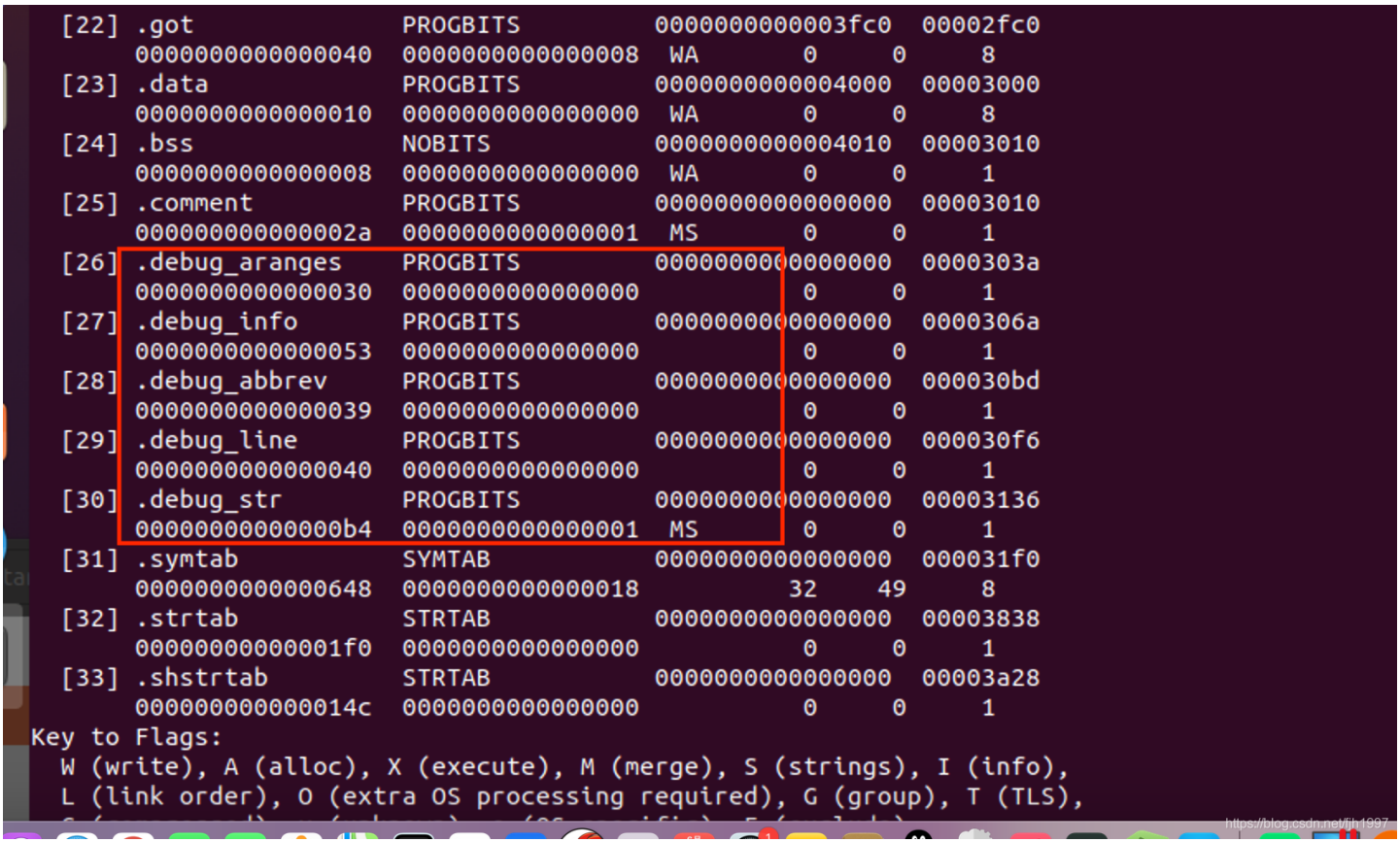
那么断点断在main函数的时候是这样的:



可以看到，gdb在加上-g参数的时候断点的时候会自动打印断点所在的源码。这是为什么呢，因为gdb 加上-g的时候编译，编译出来的elf文件会多一些段，我们成为dwarf段：

详见：https://blog.csdn.net/JS072110/article/details/44153303

我们使用`readelf -S elf`可以查看这些段。



这些带有.debug的段就是dwarf段。

那么这个时候，我们拔test.c删掉再次断在main函数会怎么样呢？

```
foo.debug  hello        hello.debug   hellodemo.debug  mama                nep.debug  test.c
ubuntu@ubuntu:~/Desktop/hello$ rm test.c
ubuntu@ubuntu:~/Desktop/hello$ gdb mama
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from mama...
(gdb) b main
Breakpoint 1 at 0x1129: file test.c, line 1.
(gdb) r
Starting program: /home/ubuntu/Desktop/hello/mama

Breakpoint 1, main () at test.c:1
1       test.c: No such file or directory.
(gdb)
```


gdb会提示找不到文件，这就说明gdb在读取程序源码文件的时候是**根据dwarf文件的信息从外部读取文件**的。这给我们利用gdb的任意读漏洞埋下了伏笔。

既然原文件名称的信息是存在dwarf文件里的，那么我们把这个文件名改为**“/flag”**会怎么样呢？岂不是就读取了题目的flag文件？

那么，如何修改elf文件里面的dwarf信息呢？这里用到了个小技巧，就是字符串替换，既然"/flag"这样一个带根目录的文件名是五个字符，那么我们也把原来的原文件命名为**五个字符**的源文件比如"nep.c"，然后编译成功后提取出debug文件，然后使用16进制编辑器将“nep.c”替换为“/flag”会怎么样呢？
替换后一尝试，就发现读取出了flag：

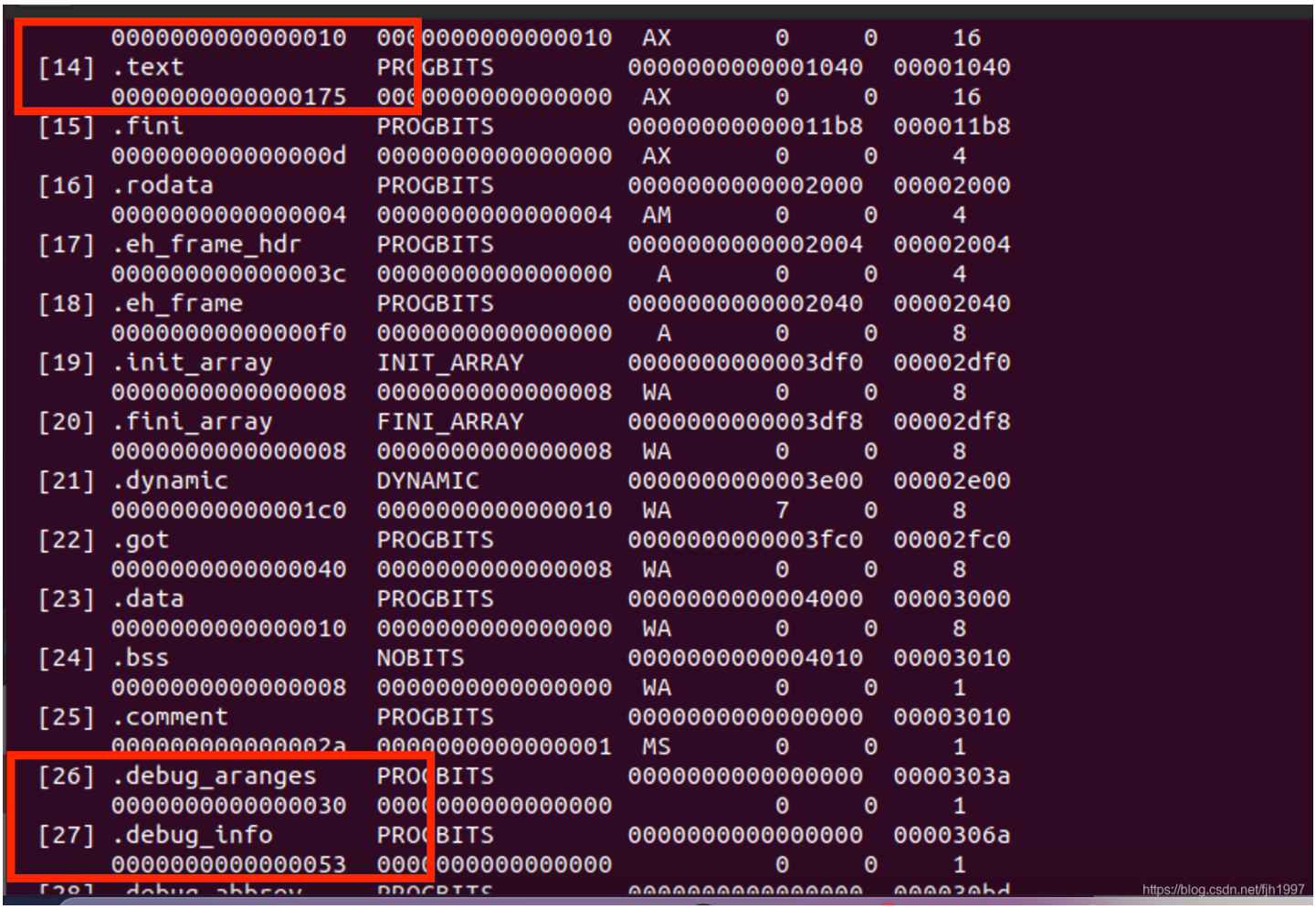
```
warning: section .plt.got not found in /home/ubuntu/Desktop/hello/hello.debug
warning: section .fini not found in /home/ubuntu/Desktop/hello/hello.debug
warning: section .rodata not found in /home/ubuntu/Desktop/hello/hello.debug
warning: section .eh_frame_hdr not found in /home/ubuntu/Desktop/hello/hello.debug
warning: section .init_array not found in /home/ubuntu/Desktop/hello/hello.debug
warning: section .fini_array not found in /home/ubuntu/Desktop/hello/hello.debug
warning: section .dynamic not found in /home/ubuntu/Desktop/hello/hello.debug
warning: section .got not found in /home/ubuntu/Desktop/hello/hello.debug

Breakpoint 1, main () at /flag:1
1       XCTF{dadadadadadadadadadadadadadada}
'a' has unknown type; cast it to its declared type
ubuntu@ubuntu:~/Desktop/hello$ ls
bak         haha2.debug  hello2.debug  hello.debug.bak  helloworld        nep.c       sy
crc32.py    haha.debug   hello.c       hellodemo        helloworld.debug  nep.debug
foo.debug   hello        hello.debug   hellodemo.debug  nep               s
ubuntu@ubuntu:~/Desktop/hello$ cat nep
```


如果文件名大于或者小于五个字符，那么由于修改dwarf信息会导致elf里面的偏移有所变化，会损坏elf文件。
当然了，这道题还有两个个比较严格的限制，
一个是debug文件必须与服务器上面正在调试的hello文件相同，这个可以用我的上一篇文章将的方法绕过：
https://blog.csdn.net/fjh1997/article/details/117415578

第二个就是
这个debug文件大小不能大于4k。由于即使一个很小的不带任何头文件的c语言编译出来的debug文件都是5，6k左右，所以我们要对elf文件进行瘦身。

经过研究，要使得这个debug文件生效，除了需要保留debug文件信息，也就是dwarf相关的段，还要保存符号表相关的段strtab .shstrtab（因为要保存main这个函数的符号表）以及.text数据段。



一开始我使用以下方法瘦身，虽然瘦了0.8k但还不够

```
objcopy --keep-symbol main --keep-symbol hello.c --strip-all   hello.debug hello2.debug #除了main函数以及源码文件hello.c的的符号表保留外，其余都删除，这样可以大大减少体积。
```

不如再彻底一点删掉所有不必要的数据段。

```
objcopy --keep-symbol main  --keep-section .debug* --keep-section .strtab --keep-section .shstrtab --keep-section .text    --only-section .symtab   nep.debug haha.debug
```

使用这样的方法对elf文件进行瘦身，即可解出题目。

最后感谢T神（ThTsOd）传授的经验和套路，爱你～～～～～～