功能：

4．隐藏被隐藏进程的TCP连接，hook /proc/net/tcp的show函数，通过被隐藏进程的/proc/pid/fd文件夹加载需要被屏蔽的tcp的inode信息，过滤这些tcp信息

功能函数：

|  |  |
| --- | --- |
| 函数 | 功能 |
| static void extract\_type\_1\_socket\_inode(const char lname[], long \*inode\_p) | 从类似于socket:[12345]中提取12345，该数字即是inode节点，否则返回-1 |
| int load\_inodes\_of\_process(const char \*name) | 检查需要被隐藏进程的fd，如果fd中存在软连接到socket的，软链接的目标将是类似socket:[12345]的形式，12345即是socket的inode节点，将这些inode节点记录下来 |
| void load\_inodes\_to\_hide(void) | 从需要被隐藏的进程中寻找需要被隐藏的socket结点 |
| char \*next\_column(char \*ptr) | 读取下一行，帮助函数 |
| int new\_seq\_show(struct seq\_file \*seq, void \*v) | 对/proc/net/tcp文件对show函数对hook函数，该文件是特殊文件，通过他可以获得tcp连接信息。调用原始函数后，对内容进行过滤，删除掉需要被隐藏的inode条目 |

原理：

1.首先调用load\_inodes\_to\_hide函数，从需要被隐藏的进程中寻找被隐藏的socket节点

2.在load\_inodes\_to\_hide函数中，需要判断是否需要被隐藏的进程，如果有，调用调用了load\_inodes\_of\_process函数把相应的inode节点记录下来。

3.在load\_inodes\_of\_process函数中，通过调用extract\_type\_1\_socket\_inode函数，判断是否含有合法的节点值。

4.hook的read函数中包含以下两句：

oldfs = get\_fs();

set\_fs(get\_ds());

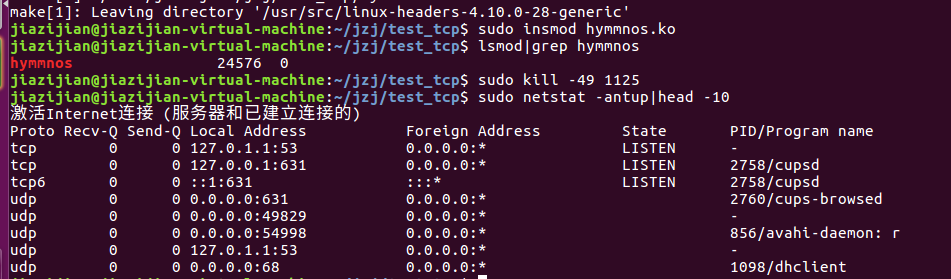
通过这这两行改变用户空间的限制，即扩大了用户空间范围，允许使用在内核中的参数了

5. 内核源码里是这样定义的，/proc/net/tcp 里的每一条记录都是 149 个字节（不算换行）长，不够的用空格补齐。所以已有量减去150即为减去一条记录的大小，也就是消除这一纪录。找到记录的起始处，在跳过tcp一些前置信息之后找到相应的inode节点，判断是否被隐藏，若隐藏则消除这条记录。

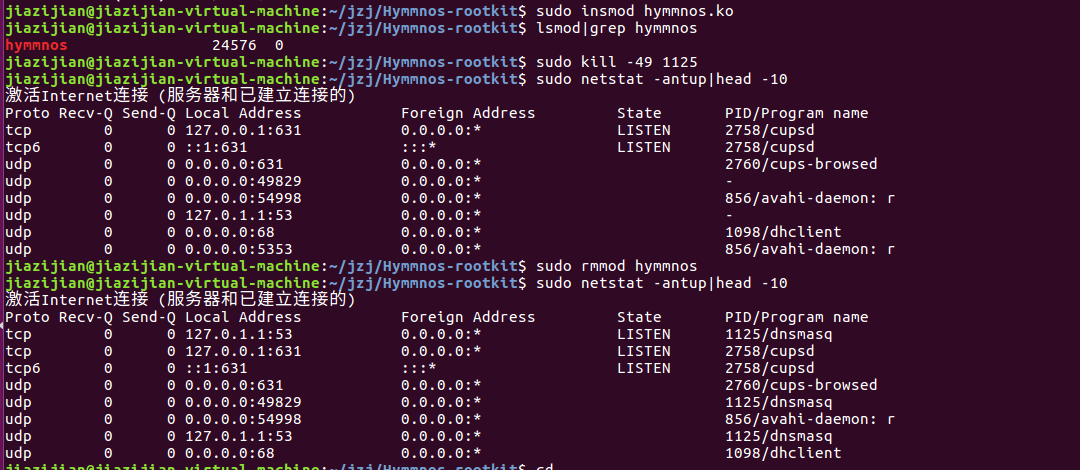
测试：

我们首先测试一下不加这个功能时系统如何表现：

我们加载一个不打在这个功能的rootkit模块，然后使用sudo netstat -antup|head -10命令可以查询到一个tcp连接的进程号为1125，我们使用隐藏进城的功能，再次调用命令，如下图所示，其中最上面的tcp连接就是进程号为1125的连接，可以看到他的进程号被隐藏了，但是整个tcp并没有被隐藏。



于是我们加入新的功能，再次加载模块得到实验结果。



可以看到这回隐藏了进城之后tcp连接也被隐藏了，当写在模块之后，被隐藏的tcp连接再次出现。

6．隐藏特定tag内的内容：

在read调用处过滤特定tag间的内容，会对文件引用尝试获取，如果大于1则说明可能是高IO文件，为不对性能产生显著影响，不做处理

功能函数：

|  |  |
| --- | --- |
| 函数 | 功能 |
| int f\_check(void \*arg, ssize\_t size) | 判断文件内容是否包含特定的tag内容 |
| int hide\_content(void \*arg, ssize\_t size) | 隐藏特定tag间的内容 |
| struct file \*e\_fget\_light(unsigned int fd, int \*fput\_needed) | 通过fd获取相应的file，降低对性能的影响 |
| asmlinkage ssize\_t new\_sys\_read(unsigned int fd, char \_\_user \*buf, size\_t count) | 对read调用的hook函数，函数开始会预先尝试获取文件的锁，失败时不做处理，这样做的原因是，需要隐藏特定内容对文件一般是不经常被读写的，所以可以获取锁，而对于高IO的文件可以降低性能影响 |

原理：

1.对read调用的hook函数，先判断调用的文件内容是否被隐藏，若未隐藏，则调用e\_fget\_light函数通过fd获取对应的文件。

2.若存在对应的文件，通过调用vds\_read函数在kernel中读取文件内容

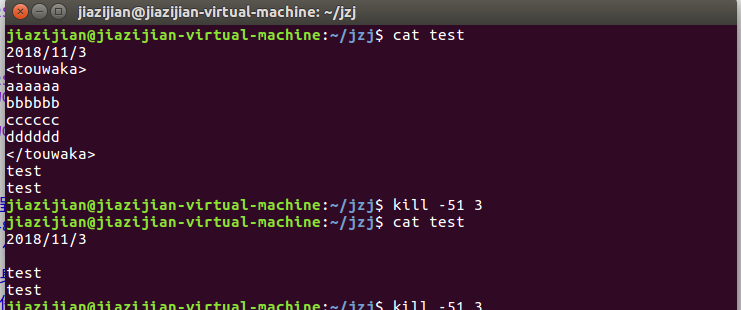
3.调用f\_check函数判断内容中是否存在包含tag的内容，若存在则调用hide\_content函数将其隐藏

4.更新文件的引用计数并返回。

测试：

这个功能通过调用read的hook函数，使用kill -51命令进行相关操作，隐藏文件中在<touwaka>与</touwaka>之间的内容，这个标签也可以在程序中自行更改。操作场景如下：

环境为已经加载了rootkit模块



可以看到原来test文件中的内容在kill -51之后中间的内容被隐藏了，当我们再次使用功能，原来的内容就会重现出来，如下图所示：

