一个拒绝服务问题的定位

秦新良

2014年 3月 9日

目录

1 问题现象

幸福的TR5总是相似的,不幸的TR5各有各的不幸。不过话说回来,没有哪个TR5是幸福的,每个版本的TR5都得“惊一番天地、泣一番鬼神。”

这次版本的TR5前,测试部开始对产品作安全性测试,测试主要针对产品的HTTPS协议作攻击测试。测试的目标是,在持续的攻击测试下,进程不会重启且正常对外提供服务。测试的场景为,四套测试集,共100W测试用例,测试集内的用例串行执行,测试集之间的用例并行执行,每个用例为一短连接;除此之外,再加上正常的业务话务。

按照上面的场景执行用例1小时后,业务进程出现拒绝服务。

2 问题定位

2.1 分析问题

进程出现拒绝服务,首先想到的是进程的资源耗尽,常见的有内存、CPU、文件描述符等。对于这种短连接的测试场景,首先想到的是文件描述符,即socket资源耗尽。登录业务单板,通过lsof命令查看进程打开的文件描述符,结果令人大迭眼镜,如下所示:

```
COMMAND     PID    USER   FD   TYPE DEVICE SIZE/OFF     NODE NAME
tcpserver 12629 vampire  cwd    DIR    8,5     4096 10879432 /home/vampire/Ubuntu One
tcpserver 12629 vampire  rtd    DIR    8,5     4096        2 /
tcpserver 12629 vampire  txt    REG    8,5    10254 10890457 /home/vampire/Ubuntu One/tcpserver
tcpserver 12629 vampire  mem    REG    8,5  1775080  3542720 /lib/i386-linux-gnu/libc-2.17.so
tcpserver 12629 vampire  mem    REG    8,5   134376  3542721 /lib/i386-linux-gnu/ld-2.17.so
tcpserver 12629 vampire    0u   CHR  136,4      0t0        7 /dev/pts/4
tcpserver 12629 vampire    1u   CHR  136,4      0t0        7 /dev/pts/4
tcpserver 12629 vampire    2u   CHR  136,4      0t0        7 /dev/pts/4
tcpserver 12629 vampire    3u  IPv4 407481      0t0      TCP *:8899 (LISTEN)
tcpserver 12629 vampire 1016u  IPv4 412846      0t0      TCP can't identify protocol
tcpserver 12629 vampire 1017u  IPv4 412847      0t0      TCP can't identify protocol
tcpserver 12629 vampire 1018u  IPv4 412848      0t0      TCP can't identify protocol
tcpserver 12629 vampire 1019u  IPv4 412849      0t0      TCP can't identify protocol
tcpserver 12629 vampire 1020u  IPv4 412850      0t0      TCP can't identify protocol
```

```
16 tcpserver 12629 vampire 1021u  IPv4 412851      0t0      TCP can't
       identify protocol
17 tcpserver 12629 vampire 1022u  IPv4 412852      0t0      TCP can't
       identify protocol
18 tcpserver 12629 vampire 1023u  IPv4 412853      0t0      TCP can't
       identify protocol
```

从 lsof 的结果可以看出,当前进程打开的文件描述符已经达到了 1023,该值是单个进程能打开文件描述符的最大值[1]。后续客户端新来的连接,由于服务端进程已没有可用的文件描述符,无法再接受这些连接,导致进程最终出现拒绝服务。

2.2 重现问题

从目前的测试场景、问题现象和已收集的信息基本可以判断进程中存在 socket 资源泄露,而且初步判断为 HTTPS 服务相关的线程中存在资源泄露。为了证实这一判断,便先把进程重启[2],执行同样的测试用例重现问题。

但很不幸,在相同的测试场景下,问题没有重现,这对定位问题来说是非常不利的。一般来说,只要是能重现的问题,基本上就不是问题了。怕就怕问题只出现一次,后面再不重现。

用例连续执行两天后,问题还是没有重现。期间通过在网上搜索资料和写模拟测试程序[3]验证,确定 lsof 的结果中出现“can't identify protocol”确实是因为服务端没有关闭 socket 导致。但通过 strace 跟踪处理 HTTPS 请求的线程,并没有发现 accept 和 close 不成对出现的情况。

在毫无头绪的情况下,便又开始重新分析问题产生前后环境上的操作日志,发现在测试套开始执行前,测试人员修改过一个软件参数,该参数也是和 HTTPS 相关的[4],但是是另一种业务的配置参数,跟本次测试的业务不相关,而且是冷配置,但测试人员在修改该参数后、执行用例前,并没有重启进程,即当时问题出现时,该参数并没有生效。

有了这上面的线索后,便试着把该参数改回默认值,重启进程,继续执行用例,结果问题竟然重现了。

[1] 虽然我们的进程提供网络服务,但实际的应用场景中,并不会有大量的连接,且单个进程限制了最大在线的连接数为 100,所以最大文件描述符使用系统的默认值 1024。

[2] 还有一个原因是,问题已经出现,光靠这一结果也是没办法定位的,只能重现出问题,找出问题必现的条件,才能进一步分析。

[3] 一个简单的 tcpserver 和 tcpclient,server 只 accept,client 端 connect 成功后立即 close。

[4] 控制业务是走 HTTPS 还是 HTTP 通道。

2.3 定位问题

问题重现了,可以说问题已经解决大半了。

通过进一步的分析发现,出现该问题的场景跟本次测试执行的用例毫无关联,这也是导致该问题难定位的重要原因,因为之前所有重现问题和分析问题的重点都放在了跟测试用例相关的场景中。那真正的场景是怎么样的?下面分别从服务端和客户端两方面来说明[5]。

首先来看服务端的连接管理。服务端对最大连接数做了限制,最大支持100个连接同时在线,但对100个之后的连接,并不会直接拒绝,而是把这些多出来的连接也接收过来,缓存到一个队列里[6](该队列默认可以缓存0xFFFFFFFE个socket),但不对这些连接做任何处理,即不会对新接收的socket上的事件做任何处理,直到前100个连接中有释放,才会从该队列中取出一个连接来处理。整个流程如图1所示。

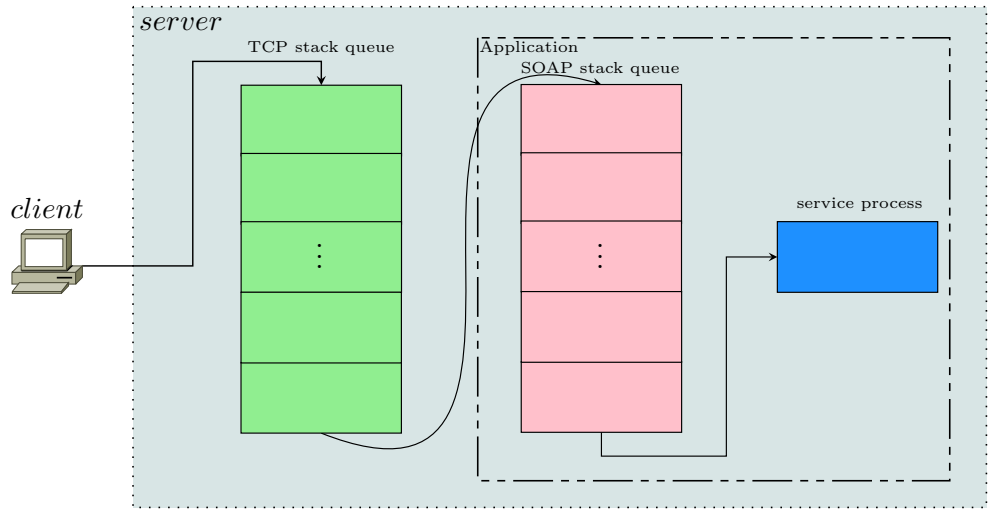

图 1: 服务端连接处理流程

客户端则是固定和服务端建立10个长连接;连接建立后,通过心跳消息来维护链路。如果心跳消息在一定时间内没有响应,就认为连接有问题,会马上将连接断连再重连。

在实际的环境配置中,客户端配置了12个进程,共需建立120个连接,这就导致有20个连接因为服务端不处理,最终会心跳超时。超时后,客户端关闭该连接后继续重连,这样反反复复,直到服务端的socket资源耗尽,出现拒绝服务(因为有其它业务也需要建立链路,但由于socket资源耗尽,连接无法建立)。

[5]一个巴掌拍不响。

[6]这么处理是为了提高HTTP短连接的性能。

3 修改方案

修改方案很简单,调整连接缓存队列大小即可。

4 问题总结

定位该问题的难点在于问题的重现,除此之外就是服务端对于底层链路的管理和维护是通过第三方库来完成的,由公司统一发布,代码不属于我们的维护范畴,这也在一定程度上对问题的定位带来了困难。

图1中,之所以把 TCP 协议栈的队列池也画出来,是因为在问题解决后的验证过程中,发现既使服务端进程的连接数已经达到最大连接数,但如果客户端继续跟服务器建链,链路还是能建的起来,通过 netstat 查看监听端口上的连接数,大于服务器进程的最大连接数,而通过 lsof 查看则没有这一现象。这是因为 TCP 的三次握手是不需要服务端进程参与的,服务进程只需要起端口监听就可以了。TCP 协议栈会自动完成三次握手,并将完成三次握手的连接放到自己的队列池中,然后如果服务进程调用 accept 的话,协议栈就从该队列池返回一个连接给服务进程。TCP 协议栈最大缓存的连接数是服务进程在起监听时调用 listen 函数时通过 backlog 参数指定的。关于该参数的说明,参见这里。