Bypassing a Next Generation Firewall

Vasileiadis A. (CyberKid)
Vasileiadis A. (CyberKid)

6 min read

Aug 20, 2024

A firewall is a critical piece of network security that acts as a barrier between trusted internal networks and untrusted external networks, such as the Internet. Monitors and controls incoming and outgoing network traffic based on predefined security rules.

Firewalls can be implemented as hardware devices, software applications or a combination of both. They serve multiple purposes:

Access Control: Firewalls regulate which network connections are allowed or blocked based on configured rules.
Threat Prevention: They protect against various external threats such as malware, viruses and potential hacker attacks.
Traffic Filtering: Firewalls can be configured to allow or block specific types of network traffic based on various criteria, including:
Source and destination IP addresses
Port numbers
Protocols
Application layer data
4. Logging and Monitoring: Many firewalls provide detailed logs of network activity, allowing security teams to detect and respond to potential threats.

5. Network Address Translation (NAT): Some firewalls perform NAT, which helps

hide internal network addresses from external networks, adding an extra layer of security.

Modern firewalls go beyond simple packet filtering. They incorporate advanced features such as deep packet inspection and application-level filtering to provide more comprehensive protection against sophisticated cyber threats.

In this article, we will focus on firewalls that operate at layer 7 of the OSI model, also known as the application layer.

Design flaw
When you look at how next-generation firewalls (NGFWs) work, you might notice something interesting. Suppose the firewall rule is “Only group A can use web ports (80 and 443)”. Initial port scanning

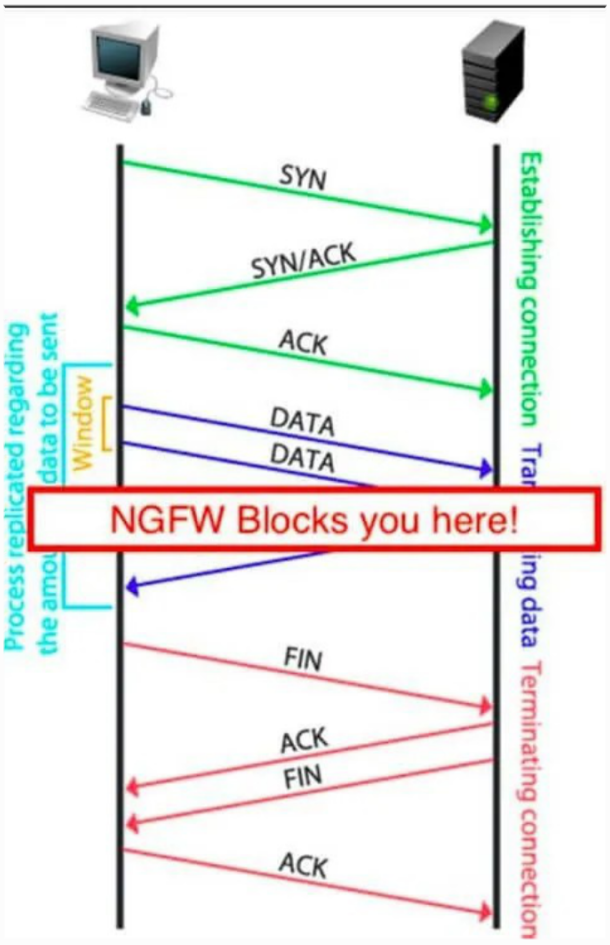
with a tool such as nmap may show numerous open ports, potentially misleading an observer into believing that the firewall is misconfigured or ineffective. However, when trying to connect to those ports that appear open, the connection drops unexpectedly soon.

A specific example of this behavior can be seen in Cisco Firepower Threat Defense (FTD), which uses the Snort engine for deep packet inspection. According to the Cisco documentation:

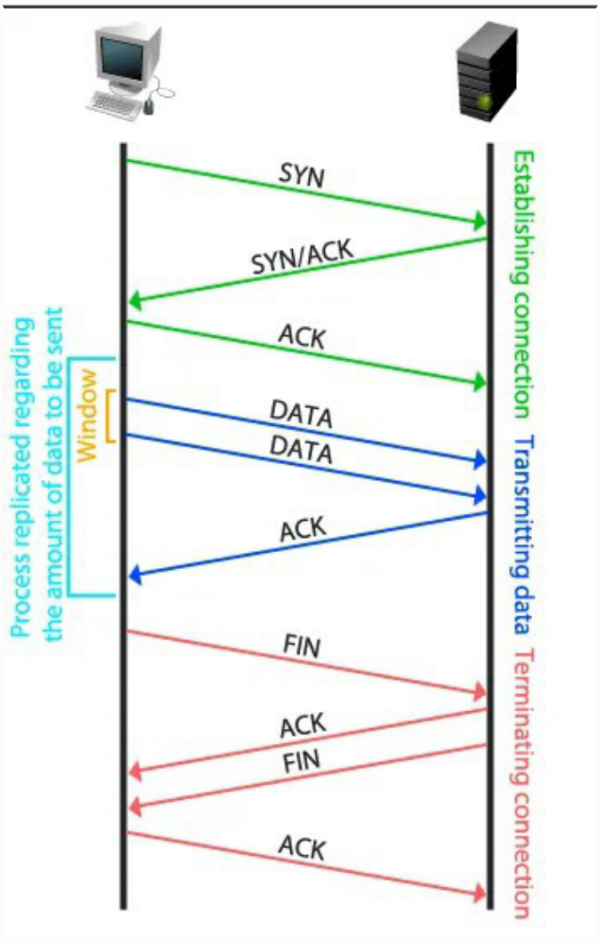
“In order for the Snort engine to determine the application, it must inspect a few packets (typically 3–10, depending on the application decoder). So some packets are allowed through FTD and reach the destination. Allowed packets are still subject to intrusion policy control based

on the “Access Policy > Advanced > Intrusion Policy used before Access Control rule is determined” option.

This means that even if a connection appears to be initiated, the firewall may block it even after parsing those initial packets. This approach allows the firewall to make more informed decisions about traffic, but it can also lead to the appearance of “open” ports that are not actually accessible.



To understand how this happens, it’s helpful to know how TCP connections work.



Establishing a TCP connection:

1. SYN: The client sends a SYN packet with a random sequence number A.

2. SYN-ACK: The server responds with a SYN-ACK. The confirmation number is A + 1 and sets its own sequence number to B.

3. Acknowledgment:The client sends an

ACK with sequence number A + 1 and acknowledgment number B + 1.

From now on, the application can start communicating and sending data through the created session. To close a connection, the endpoint that wants to terminate the communication initiates the connection termination:

Terminate a TCP connection:

1. FIN: A host sends a FIN packet to begin closing the connection.

2. Acknowledgment: The other host acknowledges the FIN.

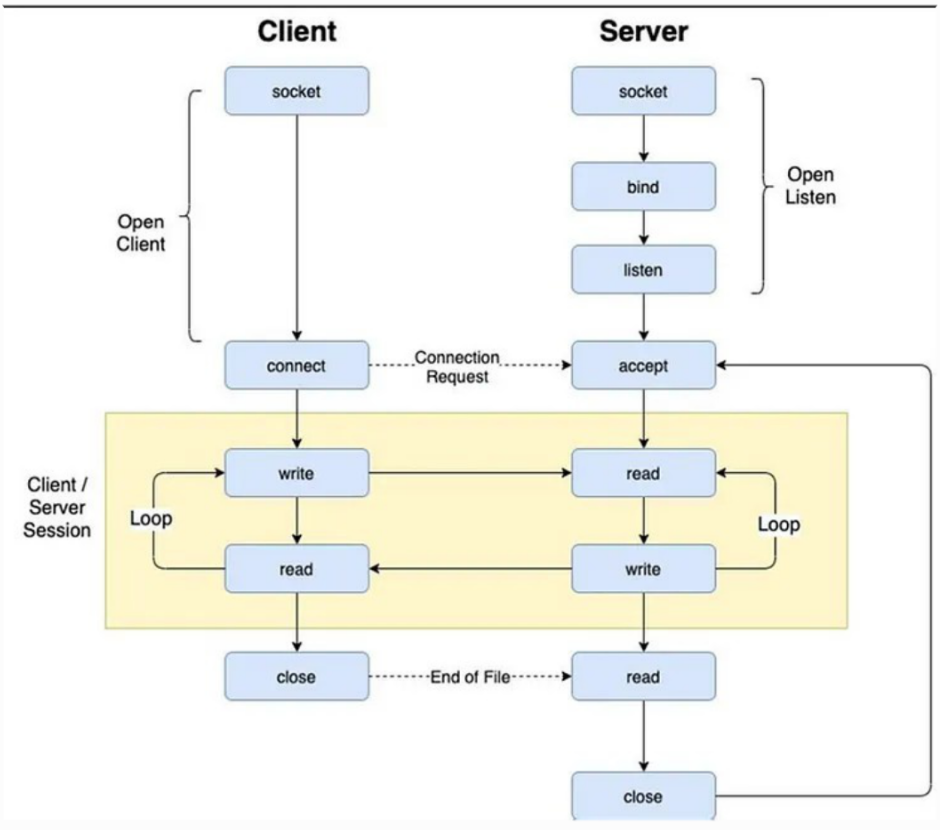
3. FIN: The second host sends its own FIN when it is ready to shut down.

4. Acknowledgment: The first host sends a final ACK.

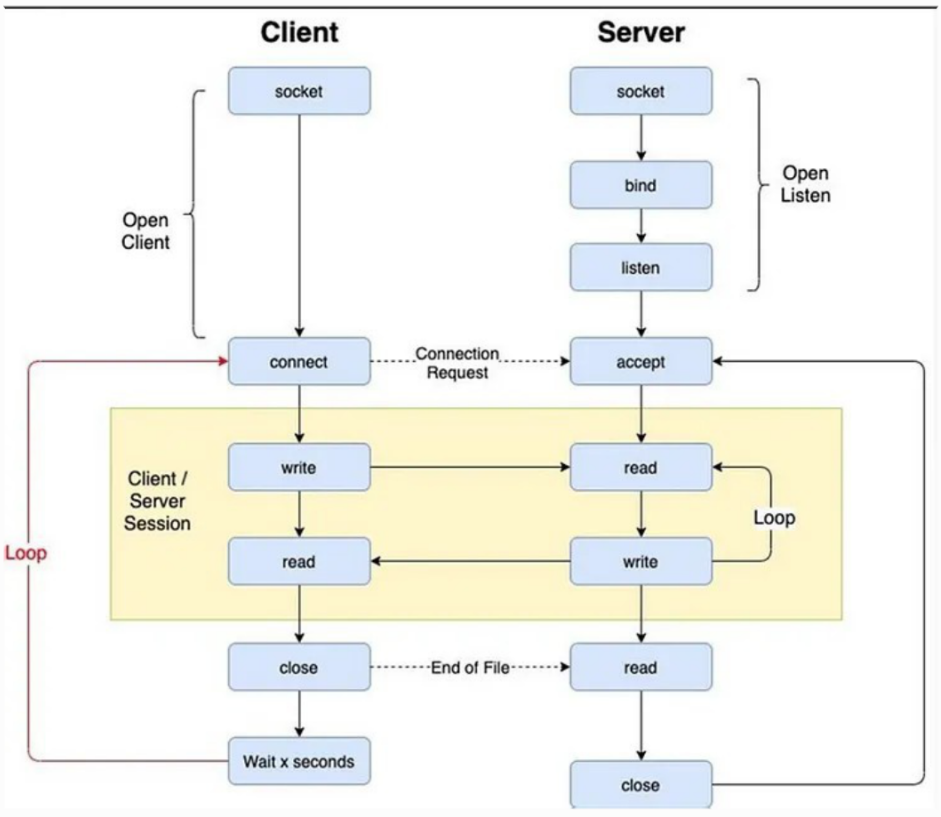
Back to Basics

Until now, we know that the IPS/IDS engine will initially let some packets through until it determines whether it is bad/malicious traffic or not. So we can change the common programming practice and take advantage of this feature.

The general architecture for socket scheduling looks like this:



We can connect, send data, receive responses, close the socket and repeat. This helps avoid detection by IDS/IPS systems that might otherwise block the connection. The modified algorithm looks like this:



That’s exactly what Fragtunnel does!

Fragtunnel
This Python-based TCP tunneling tool offers a unique method for bypassing next-generation firewalls:

Basic features:

It differs from traditional proxies or standard tunnels.
Allows application traffic to be routed to

target servers, avoiding NGFW detection.
Operating mechanism:

Data Fragmentation:The application’s incoming data is split into smaller chunks.
Multi-Session Transmission:Each fragment is sent separately through separate TCP sessions.
Data Reassembly:The fragments are recombined at the destination to reconstruct the original data.
Final Delivery:The reassembled data is forwarded to the intended target.
Installation
To get started, simply download the script from GitHub and run it. However, note that a tunnel server and a tunnel client are required to be set up for the script to work properly.

Perform the following steps on the server:

server> git clone https://github.com/efeali/fragtunnel.git

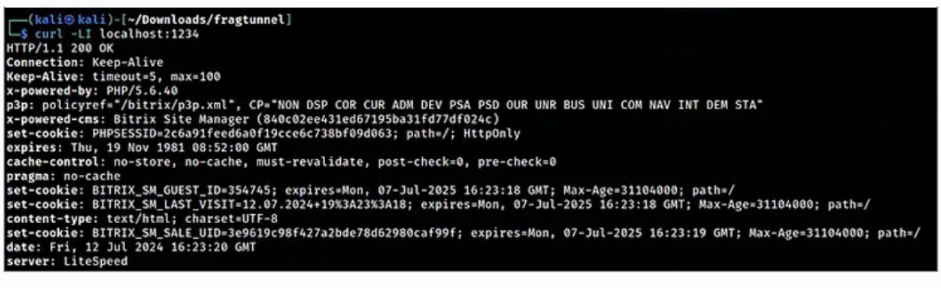
server> cd fragtunnel/

server> sudo python3 fragtunnel.py -b 127.0.0.1:80 -v

-b, –bind: Specifies the IP address and port on which the tunnel server will listen for incoming connections.
v, –verbose: Enables verbose mode, providing more detailed output or logging information when running the tunneling application.
Client side actions:

kali> sudo python fragtunnel.py -p 1234 -t :80 -T :80 -v

-p, –port:Specifies the port number on which the local application will listen to establish a connection.
-t, –target:Specifies the IP address and port of the target server or service to which the local application intends to connect.
-T, –Tunnel: Specifies the IP address and port of the tunnel server that will facilitate the connection between the local application and the target server.
Once the tunnel client executed and connected to tunnel server, you can interact with a target as if you were directly accessing it locally on your computer.



```
┌──(kali㉿kali)-[~/Downloads/fragtunnel]
└─$ sudo python fragtunnel.py -p 1234 -t [redacted]:80 -T [redacted]:80 -v
[sudo] password for kali:
Verbose mode
Local server listening on port 1234
Local connection from ('127.0.0.1', 47174)
Connected to tunnel server
Target server was set
Connected to tunnel server
Sending fragment 0
Sent data size: 78
Connected to tunnel server
Sent EOD
Connected to tunnel server
Connected to tunnel server
Connected to tunnel server
Local connection from ('127.0.0.1', 57542)
Connected to tunnel server
```

If we run tcpdump, we will notice that all communication is with the server, in this case AWS.

```
12:15:58.530323 IP [redacted] amazonaws.com.ssh > 10.0.2.15.35842: Flags [.], ack 9252, win 65535, length 0
12:15:58.550718 IP [redacted] eu-north-1.compute.amazonaws.com.ssh: Flags [P.], seq 9252:9288, ack 1, win 65535, length 36
12:15:58.550892 IP [redacted] amazonaws.com.ssh > 10.0.2.15.35842: Flags [.], ack 9288, win 65535, length 0
12:15:58.574040 IP [redacted] eu-north-1.compute.amazonaws.com.ssh: Flags [P.], seq 9288:9324, ack 1, win 65535, length 36
12:15:58.574288 IP [redacted] amazonaws.com.ssh > 10.0.2.15.35842: Flags [.], ack 9324, win 65535, length 0
12:15:58.593712 IP [redacted] eu-north-1.compute.amazonaws.com.ssh: Flags [P.], seq 9324:9360, ack 1, win 65535, length 36
12:15:58.593856 IP [redacted] amazonaws.com.ssh > 10.0.2.15.35842: Flags [.], ack 9360, win 65535, length 0
12:15:58.612505 IP [redacted] eu-north-1.compute.amazonaws.com.ssh: Flags [P.], seq 9360:9396, ack 1, win 65535, length 36
12:15:58.612662 IP [redacted] amazonaws.com.ssh > 10.0.2.15.35842: Flags [.], ack 9396, win 65535, length 0
12:15:58.633083 IP [redacted] eu-north-1.compute.amazonaws.com.ssh: Flags [P.], seq 9396:9432, ack 1, win 65535, length 36
12:15:58.633274 IP [redacted] amazonaws.com.ssh > 10.0.2.15.35842: Flags [.], ack 9432, win 65535, length 0
12:15:58.652919 IP [redacted] eu-north-1.compute.amazonaws.com.ssh: Flags [P.], seq 9432:9468, ack 1, win 65535, length 36
12:15:58.653266 IP [redacted] amazonaws.com.ssh > 10.0.2.15.35842: Flags [.], ack 9468, win 65535, length 0
12:15:58.672033 IP [redacted] eu-north-1.compute.amazonaws.com.ssh: Flags [P.], seq 9468:9504, ack 1, win 65535, length 36
12:15:58.672286 IP [redacted] amazonaws.com.ssh > 10.0.2.15.35842: Flags [.], ack 9504, win 65535, length 0
12:15:58.691676 IP [redacted] eu-north-1.compute.amazonaws.com.ssh: Flags [P.], seq 9504:9540, ack 1, win 65535, length 36
12:15:58.692018 IP [redacted] amazonaws.com.ssh > 10.0.2.15.35842: Flags [.], ack 9540, win 65535, length 0
```

Summary

In many cases, when we do reconnaissance, we notice that we can scan a server and get port information, but when we try to connect to the system, it is blocked by the firewall. A little knowledge of how these next-generation systems work can help us get past them.

The IDS/IPS engines used by most next-generation firewalls allow some data packets to reach the destination while gathering enough information to make a judgment about whether to allow or block the traffic. This is a design flaw that can be exploited to give us unfettered access to the server with a tool like fragtunnel.