```
Notes   Wireshark Filters
Global search.
tcp
udp
Only SYN flag.
SYN flag is set. The rest of the bits are not important.
tcp.flags == 2
tcp.flags.syn == 1
Only ACK flag.
ACK flag is set. The rest of the bits are not important.
tcp.flags == 16
tcp.flags.ack == 1
Only SYN, ACK flags.
SYN and ACK are set. The rest of the bits are not important.
tcp.flags == 18
(tcp.flags.syn == 1) and (tcp.flags.ack == 1)
Only RST flag.
RST flag is set. The rest of the bits are not important.

tcp.flags == 4
tcp.flags.reset == 1
Only RST, ACK flags.
RST and ACK are set. The rest of the bits are not important.
tcp.flags == 20
(tcp.flags.reset == 1) and (tcp.flags.ack == 1)
Only FIN flag
FIN flag is set. The rest of the bits are not important.
tcp.flags == 1
tcp.flags.fin == 1


TCP Connect Scan in a nutshell:

Relies on the three-way handshake (needs to finish the handshake process).
Usually conducted with nmap -sT command.

Open TCP Port   Open TCP Port
Closed TCP Port
SYN -->
<-- SYN, ACK
ACK -->
SYN -->
<-- SYN, ACK
ACK -->
RST, ACK -->
SYN -->
<-- RST, ACK

The given filter shows the TCP Connect scan patterns in a capture file.
```

```
tcp.flags.syn==1 and tcp.flags.ack==0 and tcp.window_size > 1024

TCP SYN Scan in a nutshell:

Doesn't rely on the three-way handshake (no need to finish the handshake process).
Usually conducted with nmap -sS command.
The given filter shows the TCP SYN scan patterns in a capture file.

tcp.flags.syn==1 and tcp.flags.ack==0 and tcp.window_size <= 1024

UDP Scans

UDP Scan in a nutshell:

Doesn't require a handshake process
No prompt for open ports
ICMP error message for close ports
Usually conducted with nmap -sU command.

Open UDP Port
Closed UDP Port
UDP packet -->
UDP packet -->
ICMP Type 3, Code 3 message. (Destination unreachable, port unreachable)

The given filter shows the UDP scan patterns in a capture file.

icmp.type==3 and icmp.code==3

Global search
arp
"ARP" options for grabbing the low-hanging fruits:

Opcode 1: ARP requests.
Opcode 2: ARP responses.
Hunt: Arp scanning
Hunt: Possible ARP poisoning detection
Hunt: Possible ARP flooding from detection:
arp.opcode == 1
arp.opcode == 2
arp.dst.hw_mac==00:00:00:00:00:00
arp.duplicate-address-detected or arp.duplicate-address-frame
((arp) && (arp.opcode == 1)) && (arp.src.hw_mac == target-mac-address)

Request: dhcp.option.dhcp == 3
ACK: dhcp.option.dhcp == 5
NAK: dhcp.option.dhcp == 6

"DHCP Request" options for grabbing the low-hanging fruits:
```

```
Option 12: Hostname.
Option 50: Requested IP address.
Option 51: Requested IP lease time.
Option 61: Client's MAC address.
dhcp.option.hostname contains "keyword"

"DHCP ACK" options for grabbing the low-hanging fruits:

Option 15: Domain name.
Option 51: Assigned IP lease time.
dhcp.option.domain_name contains "keyword"

"ICMP" options for grabbing the low-hanging fruits:

Packet length.
ICMP destination addresses.
Encapsulated protocol signs in ICMP payload.
data.len > 64 and icmp

"DNS" options for grabbing the low-hanging fruits:

Query length.
Anomalous and non-regular names in DNS addresses.
Long DNS addresses with encoded subdomain addresses.
Known patterns like dnscat and dns2tcp.
Statistical analysis like the anomalous volume of DNS requests for a particular
target.
!mdns: Disable local link device queries.

dns contains "dnscat"
dns.qry.name.len > 15 and !mdns

Global search
ftp
"FTP" options for grabbing the low-hanging fruits:

x1x series: Information request responses.
x2x series: Connection messages.
x3x series: Authentication messages.
Note: "200" means command successful.

---
"x1x" series options for grabbing the low-hanging fruits:

211: System status.
212: Directory status.
213: File status
ftp.response.code == 211
"x2x" series options for grabbing the low-hanging fruits:
```

220: Service ready.
227: Entering passive mode.
228: Long passive mode.
229: Extended passive mode.
ftp.response.code == 227
"x3x" series options for grabbing the low-hanging fruits:

230: User login.
231: User logout.
331: Valid username.
430: Invalid username or password
530: No login, invalid password.
ftp.response.code == 230
"FTP" commands for grabbing the low-hanging fruits:

USER: Username.
PASS: Password.
CWD: Current work directory.
LIST: List.
ftp.request.command == "USER"
ftp.request.command == "PASS"
ftp.request.arg == "password"
Advanced usages examples for grabbing low-hanging fruits:

Bruteforce signal: List failed login attempts.
Bruteforce signal: List target username.
Password spray signal: List targets for a static password.
ftp.response.code == 530
(ftp.response.code == 530) and (ftp.response.arg contains "username")
(ftp.request.command == "PASS") and (ftp.request.arg == "password")

Notes   Wireshark Filter
Global search

Note: HTTP2 is a revision of the HTTP protocol for better performance and security.
It supports binary data transfer and request&response multiplexing.

http
http2
"HTTP Request Methods" for grabbing the low-hanging fruits:

GET
POST
Request: Listing all requests

http.request.method == "GET"
http.request.method == "POST"
http.request
"HTTP Response Status Codes" for grabbing the low-hanging fruits:

200 OK: Request successful.
301 Moved Permanently: Resource is moved to a new URL/path (permanently).
302 Moved Temporarily: Resource is moved to a new URL/path (temporarily).
400 Bad Request: Server didn't understand the request.
401 Unauthorised: URL needs authorisation (login, etc.).
403 Forbidden: No access to the requested URL.
404 Not Found: Server can't find the requested URL.
405 Method Not Allowed: Used method is not suitable or blocked.
408 Request Timeout: Request look longer than server wait time.
500 Internal Server Error: Request not completed, unexpected error.
503 Service Unavailable: Request not completed server or service is down.

```
http.response.code == 200
http.response.code == 401
http.response.code == 403
http.response.code == 404
http.response.code == 405
http.response.code == 503
```

"HTTP Parameters" for grabbing the low-hanging fruits:

User agent: Browser and operating system identification to a web server application.
Request URI: Points the requested resource from the server.
Full *URI: Complete URI information.
*URI: Uniform Resource Identifier.

```
http.user_agent contains "nmap"
http.request.uri contains "admin"
http.request.full_uri contains "admin"
```

"HTTP Parameters" for grabbing the low-hanging fruits:

Server: Server service name.
Host: Hostname of the server
Connection: Connection status.
Line-based text data: Cleartext data provided by the server.
HTML Form URL Encoded: Web form information.

```
http.server contains "apache"
http.host contains "keyword"
http.host == "keyword"
http.connection == "Keep-Alive"
data-text-lines contains "keyword"
```

Global search.

```
http.user_agent
```

Research outcomes for grabbing the low-hanging fruits:

Different user agent information from the same host in a short time notice.
Non-standard and custom user agent info.
Subtle spelling differences. ("Mozilla" is not the same as "Mozlilla" or "Mozlila")
Audit tools info like Nmap, Nikto, Wfuzz and sqlmap in the user agent field.
Payload data in the user agent field.

```
(http.user_agent contains "sqlmap") or (http.user_agent contains "Nmap") or
(http.user_agent contains "Wfuzz") or (http.user_agent contains "Nikto")
```

Log4j Analysis

A proper investigation starts with prior research on threats and anomalies going to be hunted. Let's review the knowns on the "Log4j" attack before launching Wireshark.

Log4j vulnerability analysis in a nutshell:
Notes   Wireshark Filters
Research outcomes for grabbing the low-hanging fruits:

The attack starts with a "POST" request
There are known cleartext patterns: "jndi:ldap" and "Exploit.class".

```
http.request.method == "POST"
(ip contains "jndi") or ( ip contains "Exploit")
(frame contains "jndi") or ( frame contains "Exploit")
(http.user_agent contains "$") or (http.user_agent contains "==")
```

"HTTPS Parameters" for grabbing the low-hanging fruits:

Request: Listing all requests
TLS: Global TLS search
TLS Client Request
TLS Server response
Local Simple Service Discovery Protocol (SSDP)
Note: SSDP is a network protocol that provides advertisement and discovery of network services.

```
http.request
tls
tls.handshake.type == 1
tls.handshake.type == 2
ssdp
```

```
Client Hello: (http.request or tls.handshake.type == 1) and !(ssdp)
Server Hello:(http.request or tls.handshake.type == 2) and !(ssdp)
```