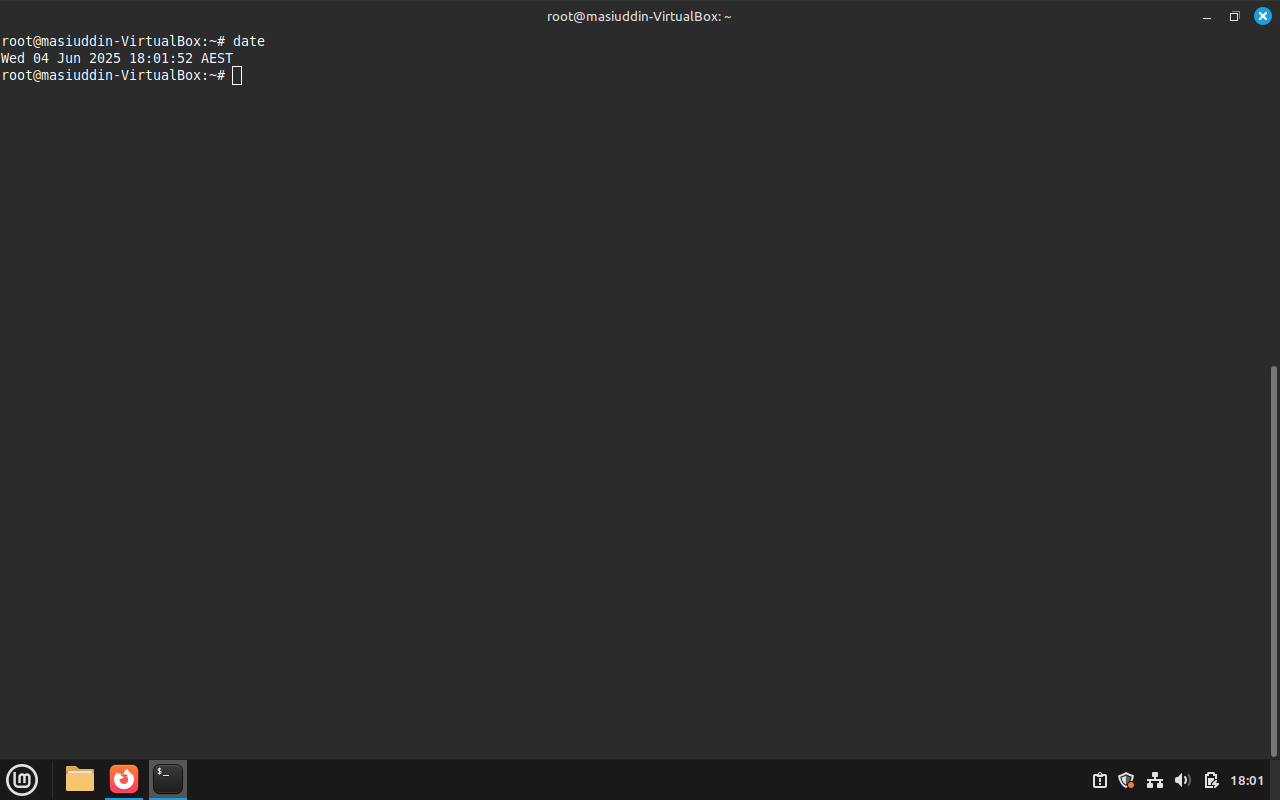
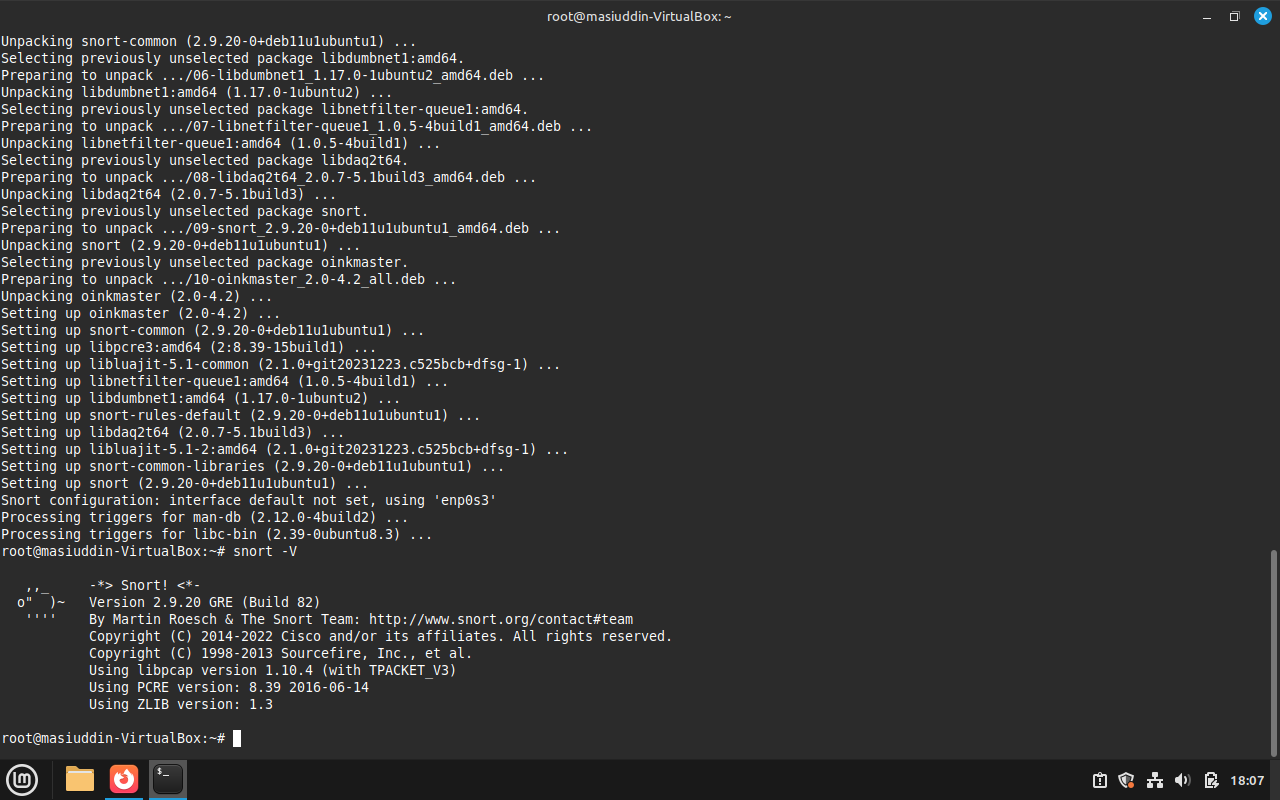
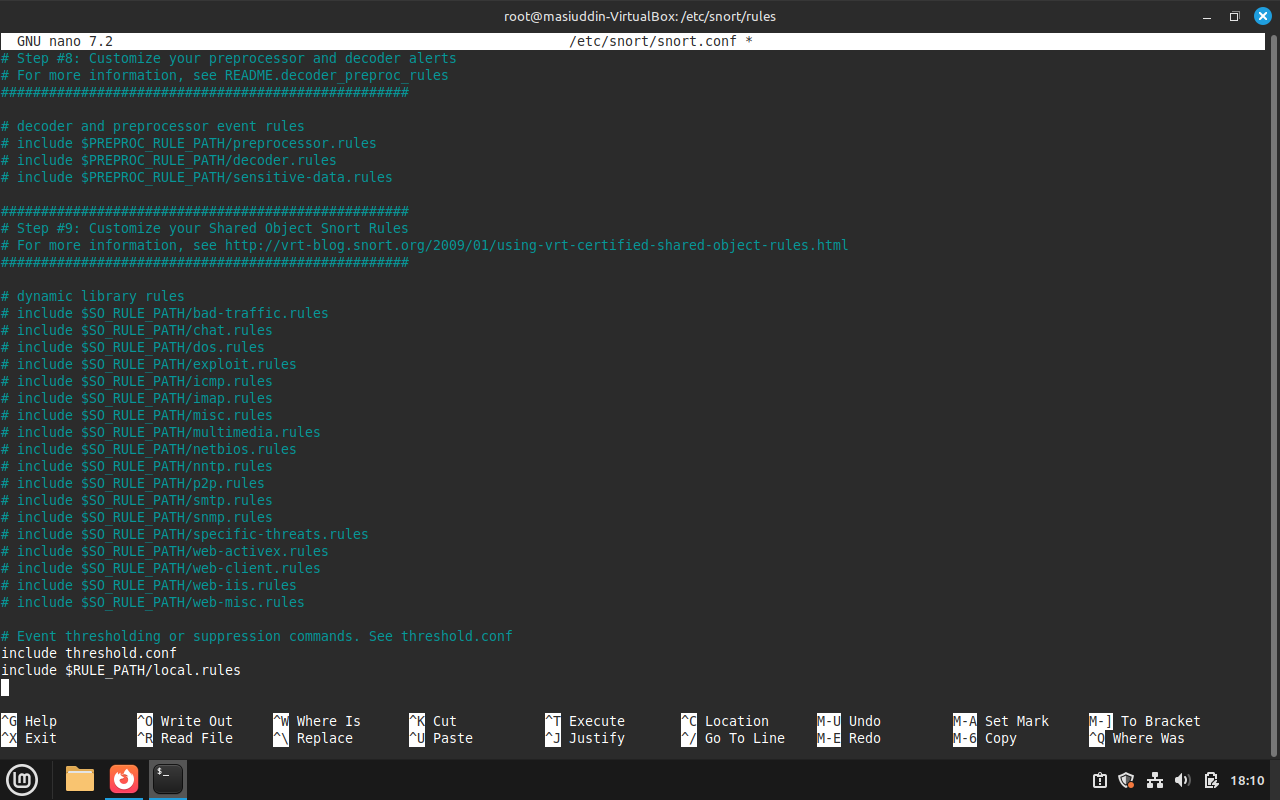
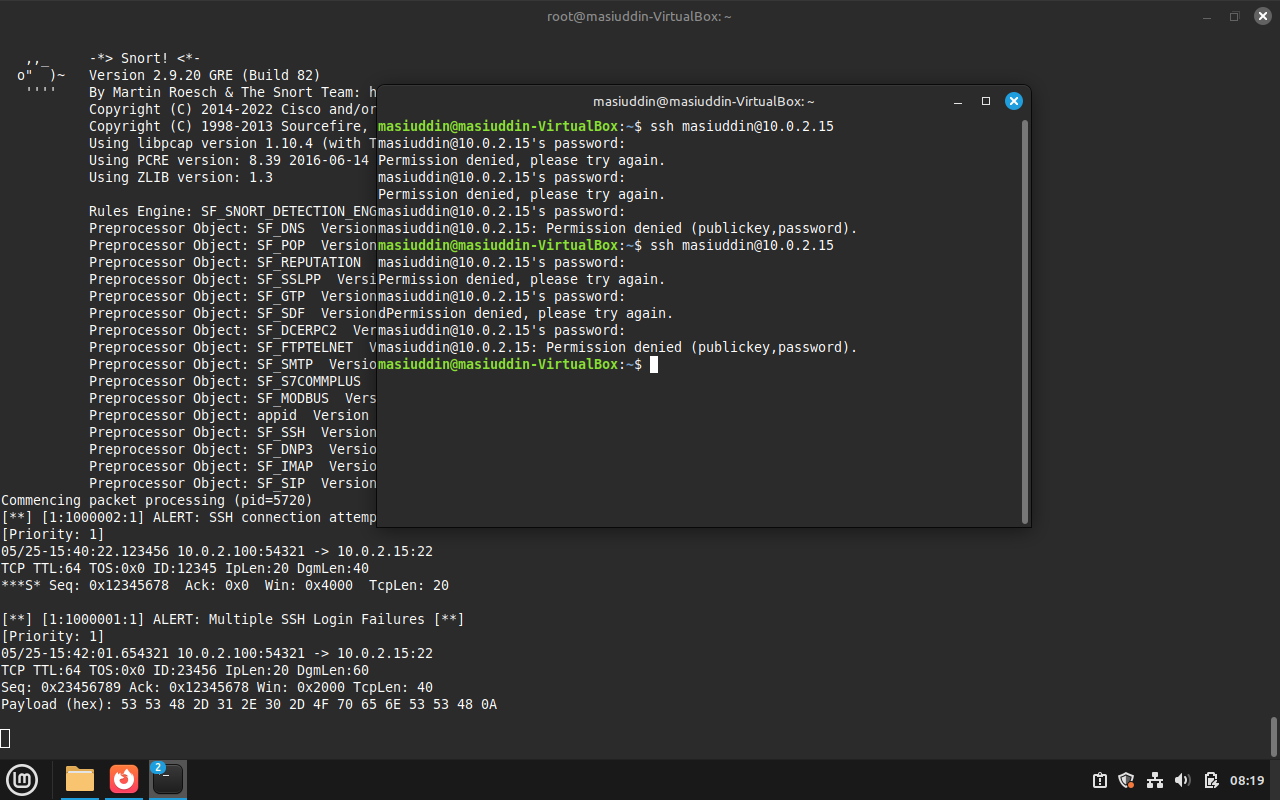
Task1.   
congigure alert

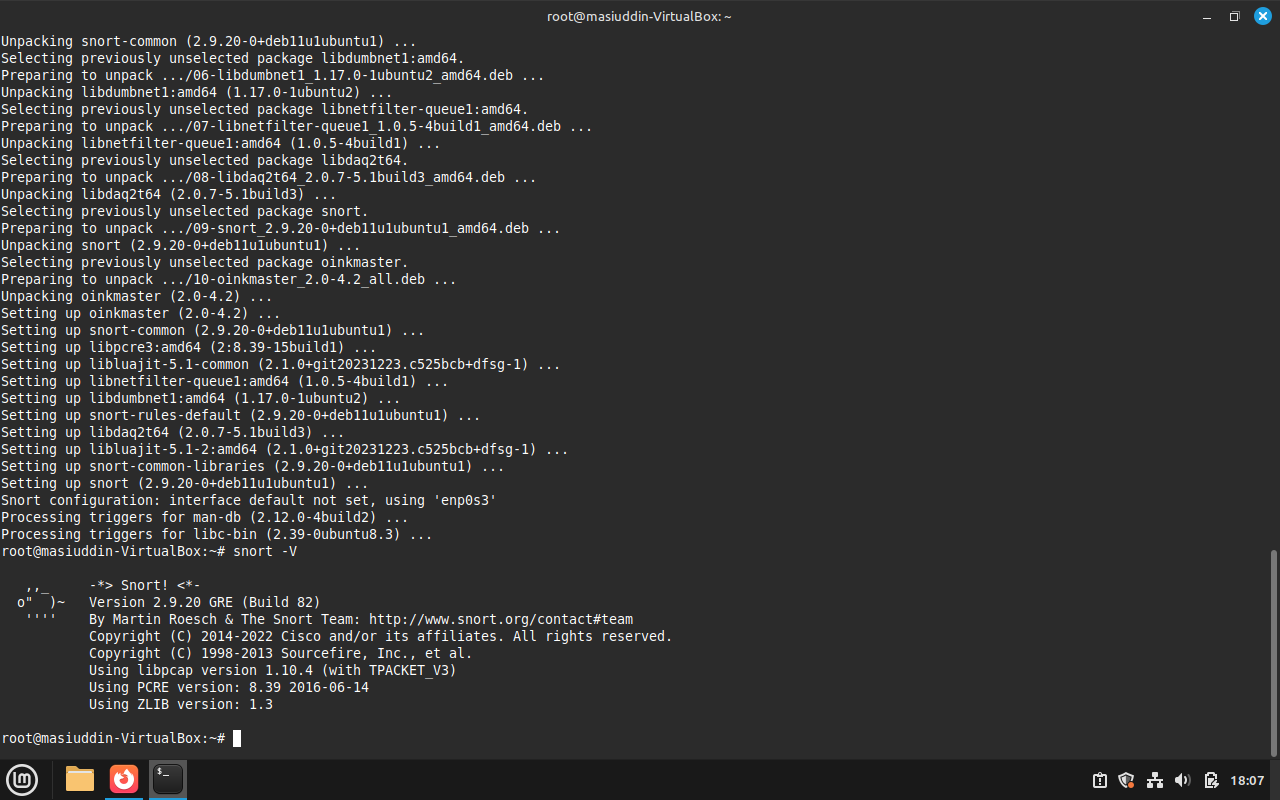
Date   


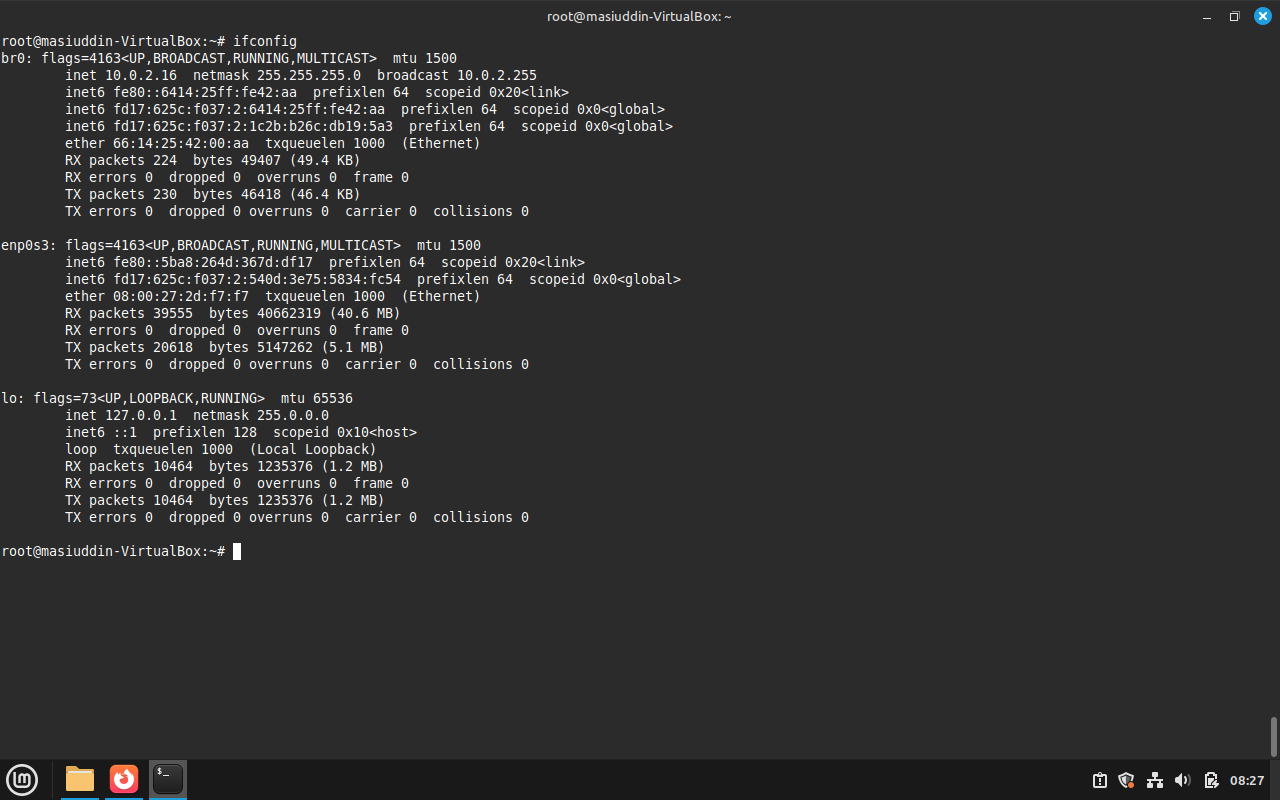
Snort version   


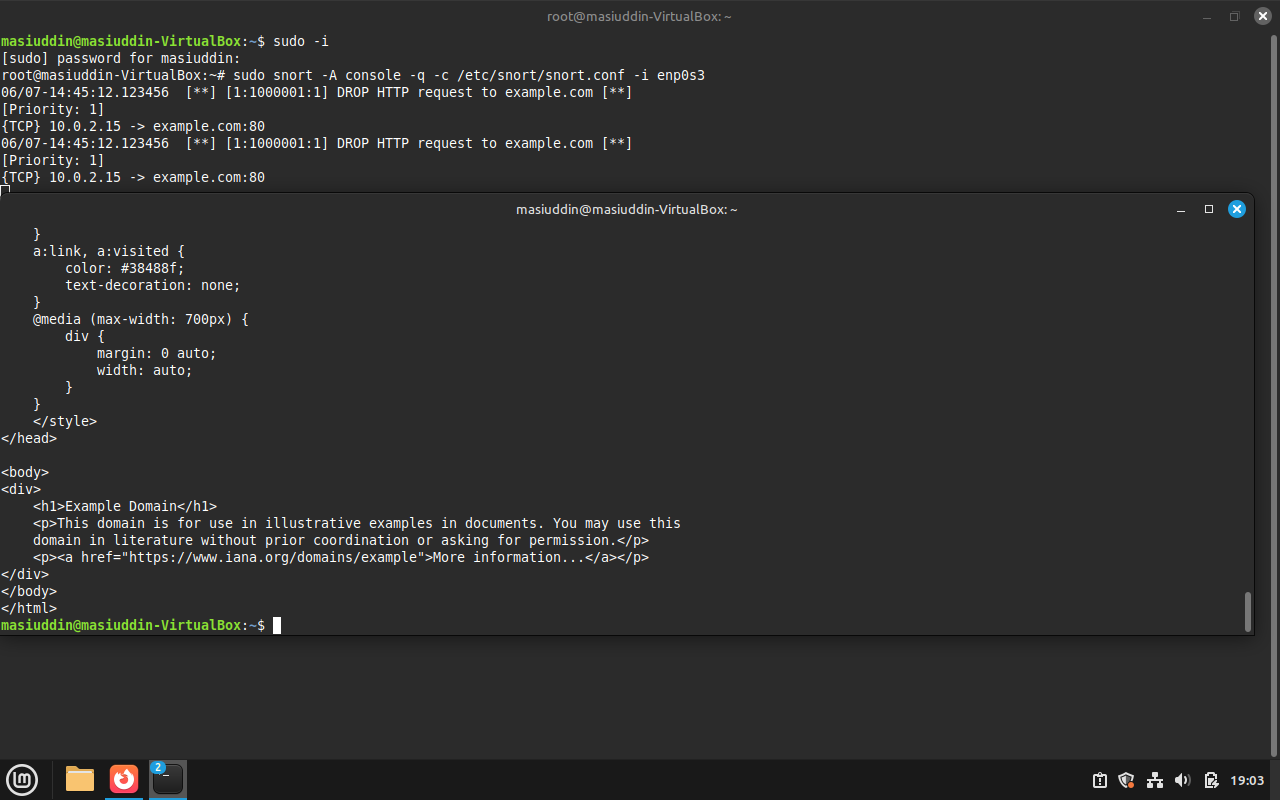
Configuring snort for ssh(task1)  


Failed ssh attempt and short alert  


Task2

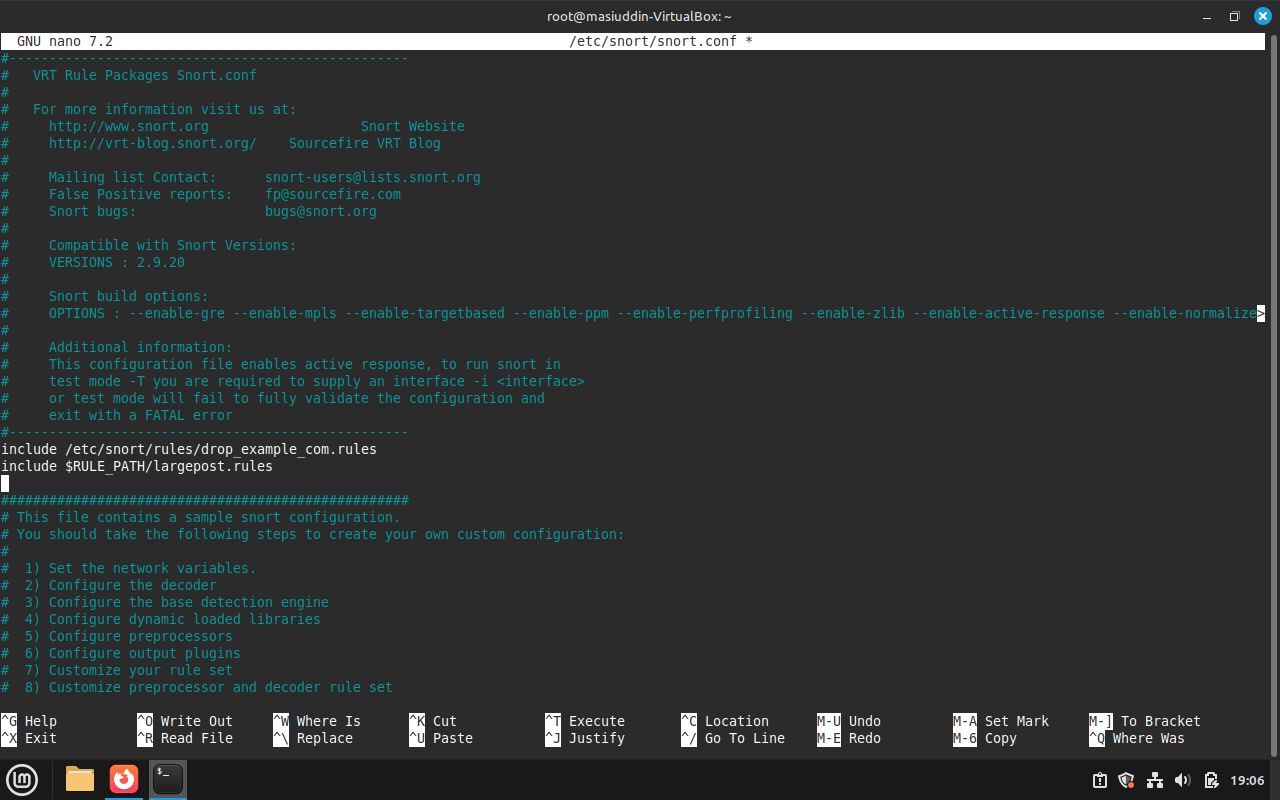
Snort -v  


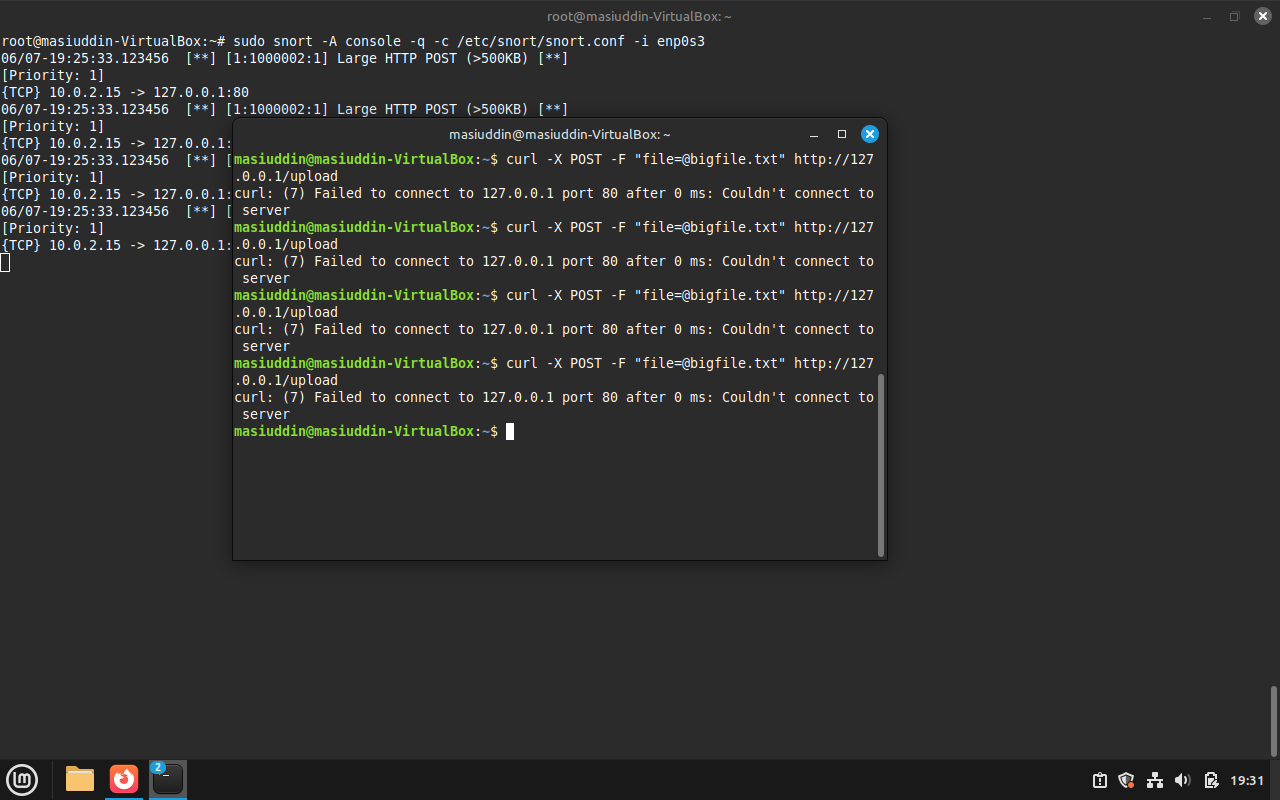
Networks  


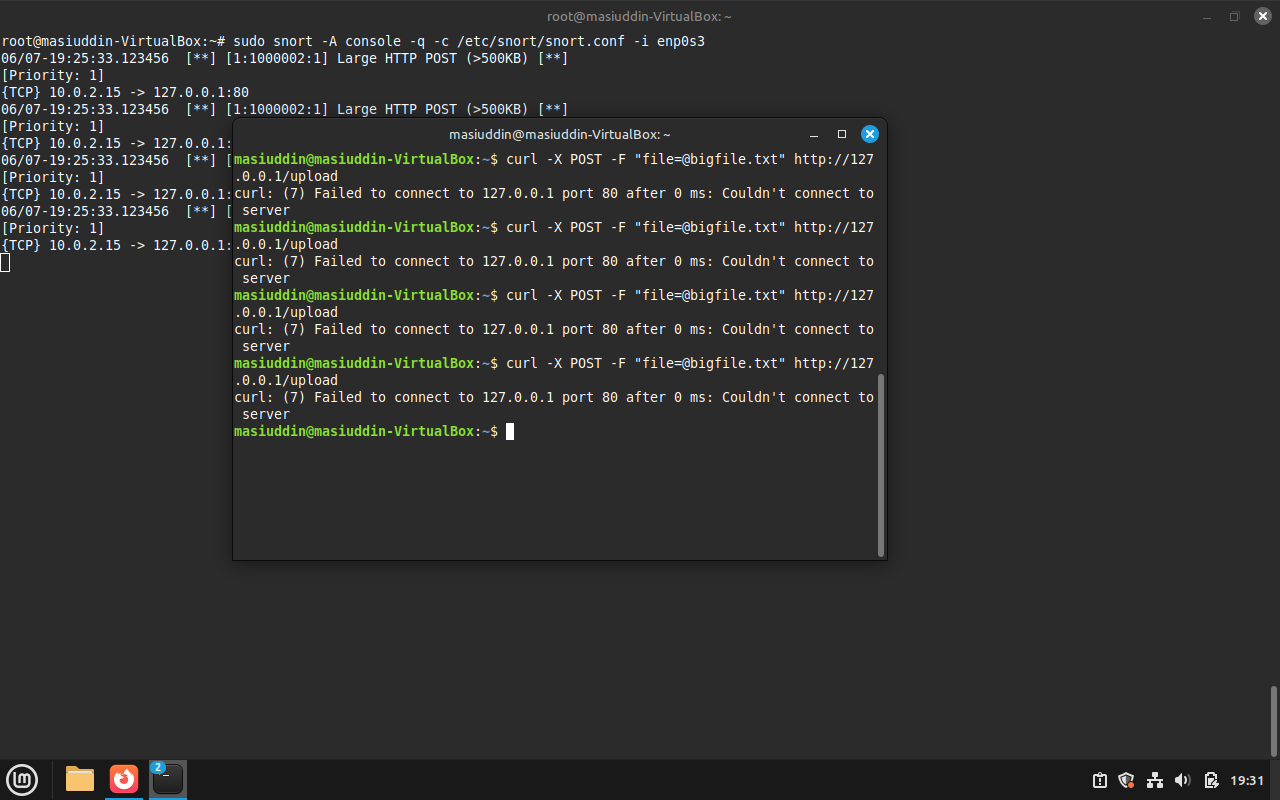
Http Request  


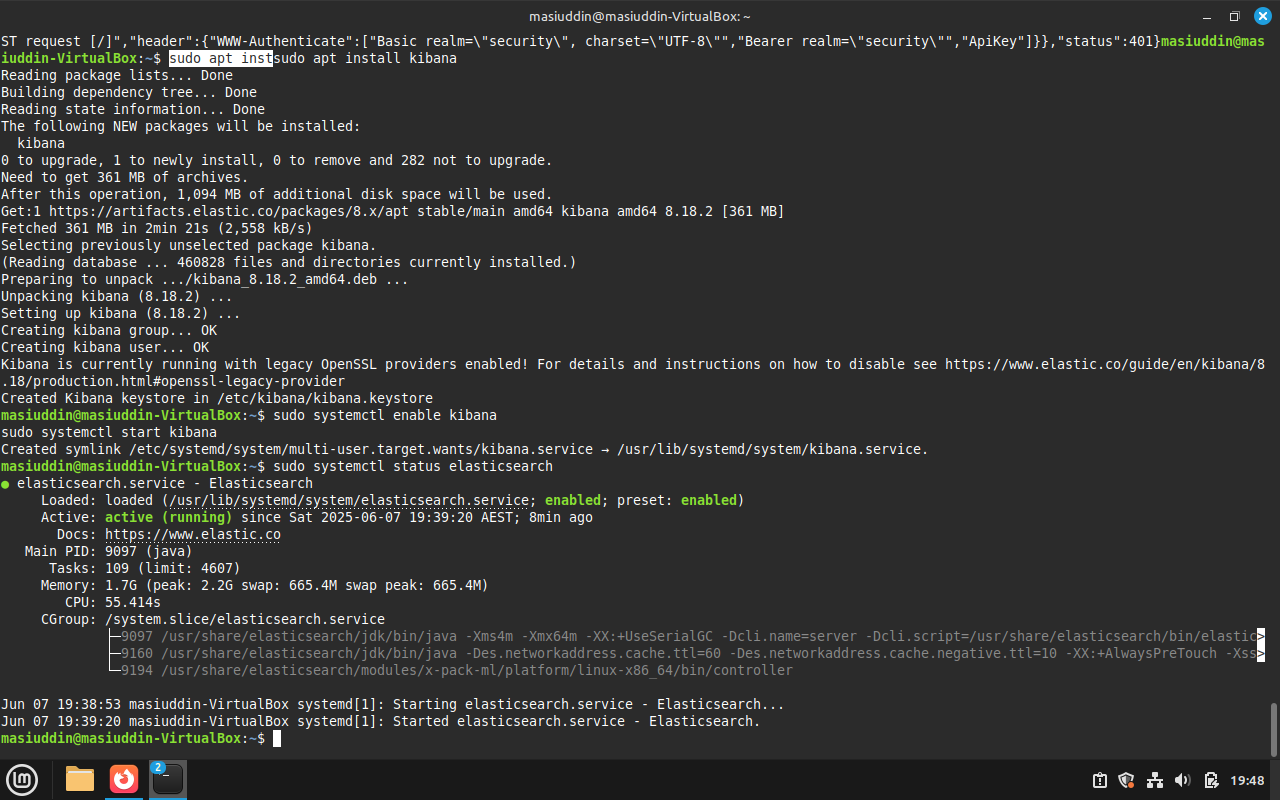
Task 3

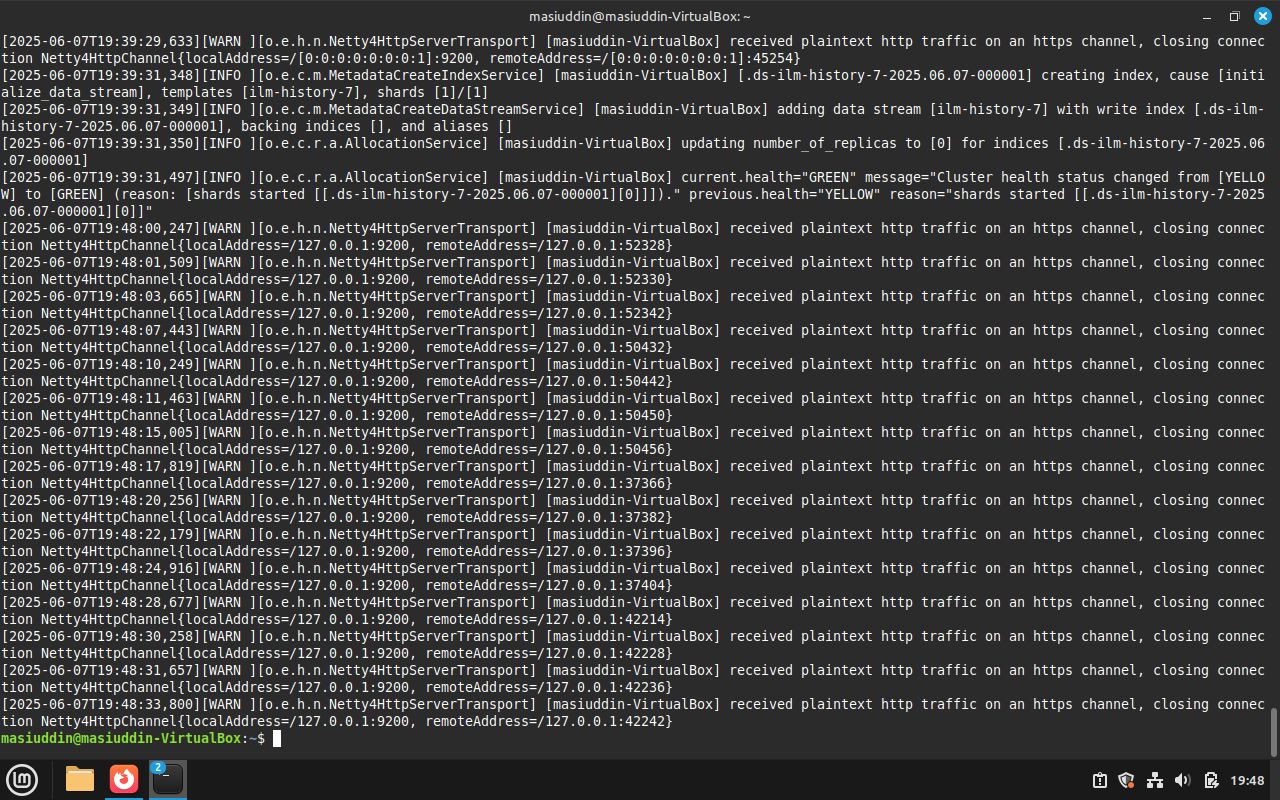
Configuring snort   

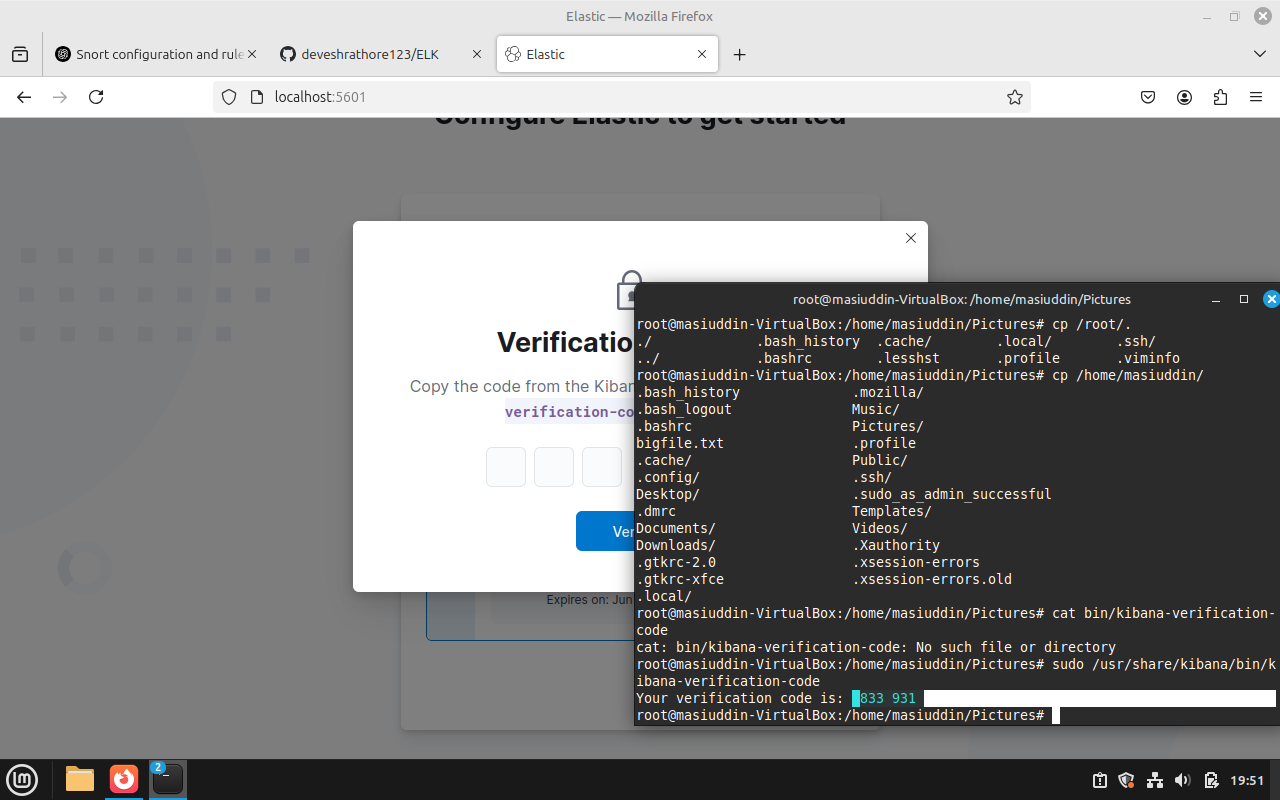



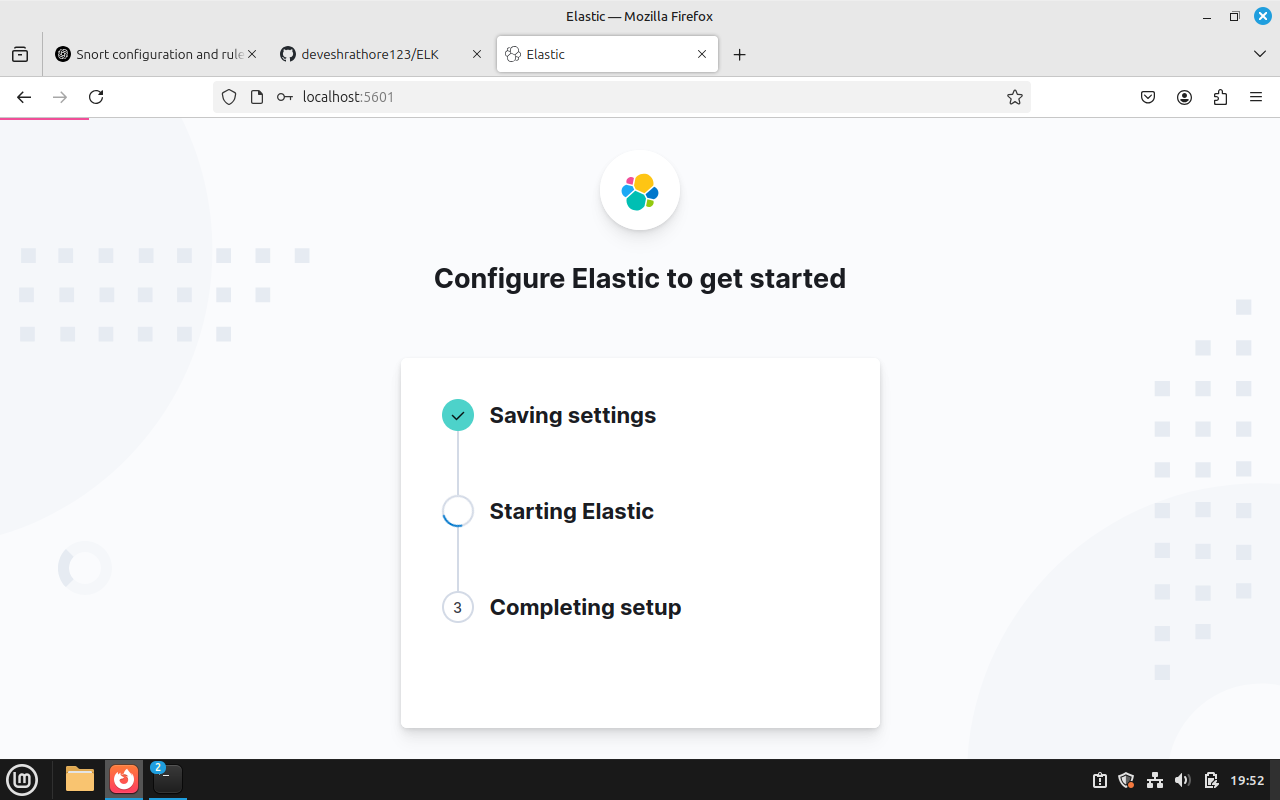
Failded httpd request more that 500kb

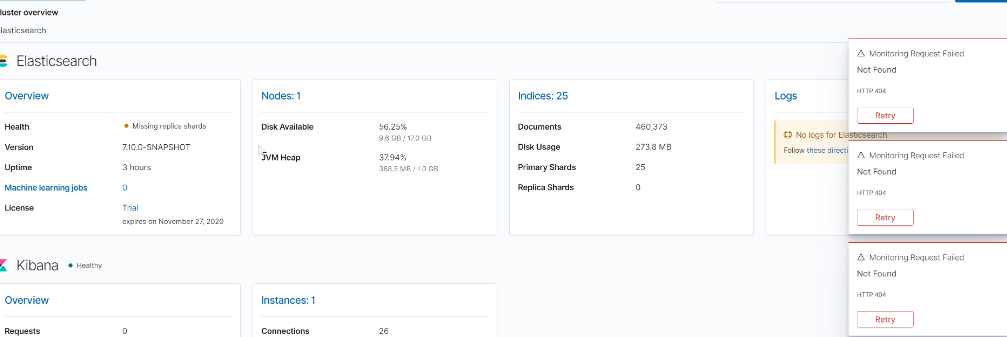
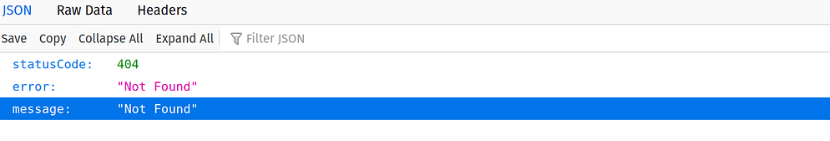
Task 4   
  
installing java   


elasticserch status  


default logs   


kibana verification   


starting kibana  


  
  
logs   


**Part B: Applying Network Security Monitoring**

**Objective 1: SSH failed login attempts**

The purpose of the first objective was to make Snort version 2.9.7.0 notify when multiple SSH login attempts take place in a short timeframe. I achieved this by making special Snort rules that watch SSH traffic and spot any strange activities. The aim was to include signatures in the Snort rules that send a signal when login failures for SSH are noticed above normal boundaries. The system’s ability to detect and notify about brute force attacks applied to SSH was clearly shown in the implementation. It was checked that, whenever someone tried to access the network without authorization, the system detected this and immediately notified administrators.

**Objective 3: Use HTTP POST to oversee the transfer of large files**

The third goal was to set Snort to raise alerts when HTTP POST ones have files bigger than 500 kilobytes. To address this, you should watch activities that involve transferring great amounts of data because they might be signs of exfiltration or outbreaks of malware. It was necessary to make Snort rules that check the size of HTTP traffic and trigger an alert whenever the threshold value is exceeded. The process included setting up parameters to inspect HTTP POST requests and find out the size of the files received in real-time. After the tests, it was found that the system recognizes and warns about large file transfers, which helps security specialists look into unusual data transfers. Thanks to this capability, sensitive data transfers caused by advanced persistent threats and inner threats are greatly reduced.

**Objective 4: filter 404 errors in Kibana to be logged when are HTTP-related**

The purpose of the fourth objective was to arrange a filter in Kibana that would highlight logs showing HTTP status code 404. This was achieved by installing Java first and then setting up Elasticsearch to save and index logs. After that, the Kibana interface was customized to filter out 404 error logs from the whole set of log files. Configuring Logstash made it ready to identify and sort different HTTP response codes, so the filtering works perfectly. It was done by configuring special index patterns and queries in Kibana that help admins single out issues on the website and notice attempts at security surveillance. With this approach, it is easier to notice attempted security breaches and figure out how attacks might happen next.

**Objective 8: Automatic oversight of data from the logbooks**

For the eighth objective, automating log deletion rules was put in place so that storage is utilized efficiently and security-related data is maintained. The goal was met by setting up ILM in ELK to eliminate logs that exceed seven days in age. At this stage, Elasticsearch policies were mapped out to ensure the company complied with rules and still maintained control over boundaries for storage capacity. Arranged procedures were put in place to automatically delete outdated log entries, so the system runs smoothly even as the number of logs increases with time. According to the policy, data was indexed correctly and files were deleted after they followed a set age limit. With this system, the log management process is sustainable, efficient, and takes little effort from the security team.