Journal of Engineering Research and Education
Volume 18, (Special Issue) 2026 [149-162]

A Low-Cost Traffic Generation System Using Open-Source Software for Accessible Cyber Ranges

Zainab S. Attarbashi[1], Auni Fatini Saniza[1], Nazihah Shafudin[1], Oula Sakka[2], Atikah Balqis Basri[2], and Azana Hafizah Mohd Aman[3]

[1]Department of Information Systems, Kulliyyah of Information & Communication Technology (KICT), International Islamic University Malaysia, Gombak, Malaysia
[2]Department of Computer Science, Kulliyyah of Information & Communication Technology (KICT), International Islamic University Malaysia, Gombak, Malaysia
[3]Faculty of Information Science and Technology, Center for Cyber Security, Universiti Kebangsaan Malaysia, Bangi, Malaysia

ABSTRACT

This project implements a comprehensive open-source traffic generator designed for cyber ranges and network defence labs. The project addresses the growing demand for advanced cybersecurity training tools that can simulate real-world network traffic and attack scenarios. By providing a hands-on laboratory environment, the tool directly enhances cybersecurity education by allowing students to interact with and analyse dynamic network conditions and malicious traffic patterns. Traditional traffic generators often lack the flexibility and usability required for educational and research applications, creating a gap that this project seeks to fill. The developed traffic generator features an intuitive configuration and control interface, a robust traffic generation engine capable of simulating diverse network traffic patterns, and support for various protocols, including TCP, UDP, ICMP, and HTTP. Prepackaged attack scenarios, such as DDoS, ransomware, and reconnaissance, enhance realistic cybersecurity training. The tool prioritizes customization options and comprehensive reporting to maximize its functionality and adaptability to different use cases. By offering an effective and accessible solution, this project significantly enhances the training and preparedness of cybersecurity professionals, contributing to the development of secure and resilient network infrastructures.

Keywords: Cyber Range, Cybersecurity, Traffic Generator, Open-source Tools, Prototype Implementation

1. INTRODUCTION

The increasing complexity and frequency of cyber threats have highlighted the need for advanced training tools to prepare cybersecurity professionals effectively. Cyber ranges and network defence labs play a crucial role in providing hands-on training by simulating real- world attack scenarios [1]. However, traditional traffic generators often lack the flexibility and usability required to create realistic and customizable training environments [2], [3]. This restriction makes it more difficult for network defence and cybersecurity laboratories to offer thorough training settings [4]. As shown in Table 1, a significant gap exists between the capabilities of traditional tools and the requirements of modern cybersecurity education. Aspiring cybersecurity experts could find it difficult to obtain hands-on experience in recognizing and addressing sophisticated threats in the absence of easily navigable and customized tools[5].

*zainab_senan@iium.edu.my

Table 1 Limitations of Existing Traffic Generators for Educational Use

Feature	Traditional/ Commercial Generators	Requirement for Effective Training	Impact of the Gap
Cost	High licensing fees	Low-cost / Open-Source	Limits access for universities and SMEs.
Usability	Complex, CLI-heavy interfaces	User-Friendly & Accessible	Steep learning curve distracts from core training objectives.
Customization	Pre-defined, rigid traffic profiles	Highly Customizable Scenarios	Fails to adapt to specific learning outcomes or new threats.
Realism	Often simplistic or repetitive patterns	Realistic, Multi-Stage Attack Flows	Trainees are not prepared for the dynamism of real attacks.
Educational Scaffolding	None	Integrated Hints & Progress Tracking	Lacks support for guided learning and skill assessment.

The goal of this project is to research and assess current traffic generators and network defence labs to identify strengths and create a unique traffic generator tailored for network defence labs or cyber ranges, specifying hardware-software interactions/ and packet generation methods. In addition, the traffic generator will be tested, its performance, and its effectiveness in simulating cyber threats will be improved. The solution will be improved based on feedback and comparative analysis with existing tools. As a result, it will provide a cost-effective and scalable solution for cybersecurity training.

1.1 Existing Traffic Generator Tools

A customized program called the synthetic network trac generator (TG) is made to mimic network activity within predefined bounds [6]. They are essential to training, testing, and research, among other facets of cybersecurity [7]. Traffic generators such as Distributed Internet Traffic Generator (D-ITG), Cisco TRex, and Ostinato were analyzed for their key features, advantages, and limitations.

The Distributed Internet Traffic Generator (D-ITG) is a platform that can generate traffic that precisely follows patterns determined by the packet size (PS) and inter-departure time (IDT) of stochastic processes. The Inter Departure Time (IDT) and Packet Size (PS) of real applications are emulated by D-ITG using stochastic processes. It supports multiple statistical distributions for the IDT and PS random variables (exponential, uniform, cauchy, normal, pareto, etc.). The host executing the traffic generator controls how the host manages time (process scheduling, time function resolution, etc.), making the IDT the more susceptible to inaccurate data of the two random variables [8]. The generation loop, which is repeated for each generated packet, shows that different memory accesses (mem), system calls (sys), CPU computations, and I/O requests are contained in it [8]. This shows the operation of packet generations.

Trex [9] is an open-source traffic generator developed by Cisco Systems which can be used in a stateless and stateful configuration. When in stateless mode allows for the creation of several packet generation streams, the alteration of any field inside the packet (headers, trailers, payload, flags, etc.), and even the intentional delivery of corrupt packets to gauge the network's reaction [10]. With specialized hardware, including network interface cards (NICs) that enable hardware-based packet processing, TRex can create packets at wire speed. As a result, TRex can generate traffic with a high throughput and minimal latency. The traffic generation engine that powers TRex is installed on a workstation or high-performance server. Network traffic is generated by the engine using user-specified parameters and profiles.

Another open-source packet crafter and traffic generator is Ostinato where users can make, modify, and broadcast unique network packets using an open-source packet crafter and traffic generator [10]. For the purpose of generating and modifying network packets at different OSI model layers, such as Ethernet, IP, TCP, UDP, ICMP, and others, Ostinato offers an intuitive graphical user interface. To create unique packet payloads, users can designate packet attributes such source and destination addresses, protocol headers, payload data, and checksums. The automation and scripting features of Ostinato allow users to design scenarios and traffic patterns. This enables users to simulate diverse network conditions and behaviors by creating traffic with varying characteristics, including packet size, interarrival time, rate, and distribution.

1.2 Advantages of Using Traffic Generator Tools

Distributed Internet Traffic Generator (D-ITG), TRex, and Ostinato are all the most used traffic generators with their own specialities and capabilities. One way that D-ITG sets itself apart is through its distributed design, which enables traffic to be generated by coordinating several instances across various nodes. This distributed method improves load balancing and scalability, which makes it appropriate for testing extensive network deployments. Furthermore, D-ITG supports many traffic generation models and protocols including TCP, UDP, ICMP, DNS, Telnet, and VoIP (G.711, G.723, G.729, Voice Activity Detection, Compressed RTP), allowing users to generate a wide range of traffic patterns for extensive testing scenarios. TRex by Cisco System has high-performance traffic generation capabilities, making it ideal for line-rate testing of network devices and applications. Testing high-speed networking equipment requires a high throughput and low latency, which TRex provides by utilizing hardware-based packet processing. Because of its stateless operation, which ensures accurate performance metrics and realistic traffic patterns, it can generate traffic efficiently without incurring the burden of keeping session state.

Ostinato [11] is notable for its flexible packet construction and traffic generating capabilities [12], as well as its user-friendly graphical interface. Both inexperienced and seasoned users may construct and alter bespoke network packets with ease because of its user-friendly architecture. Additionally, it supports an abundance of protocols, giving the freedom to customize packets and stack protocols [10]. Ostinato is appropriate for a variety of testing and experimental jobs due to its broad protocol support, which includes Ethernet, IP, TCP, UDP, and more. The selection of these tools is dependent upon the particular demands and goals of the experimentation or investigative activities [13][14], as each presents unique benefits customized for diverse uses. To summarize the key characteristics of the analyzed tools, Table 2 provides a comparative overview based on their primary strengths, operational modes, and ideal use cases.

2. METHODOLOGY

The aim of this project is to provide a complete and user-friendly open-source traffic generator designed especially for cyber ranges and network defense labs. Our technique is methodical and is broken down into discrete stages to guarantee a complete and efficient development process. To determine the precise requirements and limitations of the targeted users, a thorough requirement gathering was made first (see table 3 for the requirement specifications). The next step is the design process, during which a suitable open-source tools and frameworks were explored, and a comprehensive architectural design was drafted. Using reliable open-source libraries like DPDK, the real coding and integration of the traffic generator takes place during the implementation phase. Subsequently, extensive testing is carried out to verify the traffic generator's performance, security, and usefulness.

Table 2 Comparison of Open-Source Traffic Generation Tools

Feature	D-ITG	Cisco Trex	Ostinato
Primary Strength	Distributed testing & precise traffic emulation	High-performance, line-rate traffic generation	User-friendly GUI for custom packet crafting
Traffic Modeling	High-fidelity; uses statistical distributions (Exponential, Pareto, etc.) for Inter-Departure Time (IDT) & Packet Size (PS)	Stateless & stateful modes; can modify any packet field and inject corrupt packets	Customizable packets with defined size, rate, inter-arrival time, and distribution
Key Advantage	Supports a wide range of protocols (TCP, UDP, ICMP, DNS, VoIP) for comprehensive scenario testing.	Leverages hardware (NICs) for wire-speed performance with high throughput and low latency.	Intuitive graphical interface and deep protocol support (Ethernet, IP, TCP, UDP, etc.) for easy use.
Operational Mode	Distributed (coordinates multiple instances across nodes)	Centralized (high-performance server/workstation)	Centralized (desktop application)
Ease of Use	Moderate (Command-line interface)	Complex (Requires scripting and understanding of high-performance networking)	Easy (Graphical User Interface)
Ideal Use Case	Research and testing requiring precise, distributed traffic patterns across a network.	Stress-testing and performance validation of high-speed network devices and infrastructure.	Educational labs, protocol testing, and crafting custom packets for specific scenarios.
Core Limitation	IDT accuracy can be susceptible to host system load and scheduling.	Steep learning curve; requires specialized hardware for maximum performance.	Not designed for generating line-rate, high-volume traffic.

2.1 System Development Process

The system was implemented by setting up the Cisco TRex traffic generator inside one of the PCs in the cyber range lab, followed by generating network traffic from the traffic generator to other PCs in the lab. The overall flowchart of this process is illustrated in Figure 1. The traffic generator was installed in the Kali Linux Virtual Machine that is hosted by a Proxmox-based virtualized environment. Proxmox Virtual Environment (Proxmox VE) is an open-source server virtualization platform built on Debian Linux, where it facilitates the development and deployment of virtual machines (VMs) and containers by effectively managing computational resources. Wireshark is also used to confirm that the traffic generated by T-Rex matches the specified configuration such as the source and destination IPs, packet size, traffic rate, packet type, and duration. The development began with the implementation of core functionalities, such as the traffic generation engine and basic protocol support, followed by the addition of supporting features like reporting, customization options, and scalability. Each phase included rigorous testing and validation to ensure accuracy, performance, and reliability.

Table 3 Requirements Specification Table

No.	Feature/ Task	Description	Priority Level	Rationale
1	User Interface Design	Develop a user-friendly interface for configuring and controlling the traffic generator.	High	Essential for ease of use and accessibility by instructors and students.
2	Traffic Generation Engine	Implement the core functionality to generate various network traffic patterns.	High	Fundamental component that drives the simulation of cyber threats.
3	Protocol Support	Support a wide range of network protocols (e.g., TCP, UDP, ICMP, HTTP).	High	Necessary to mimic realistic network traffic and diverse attack scenarios.
4	Attack Simulation	Implement predefined attack scenarios (e.g., DDoS, malware, reconnaissance).	High	Key feature for cybersecurity training and testing.
5	Customization Options	Allow users to customize traffic parameters such as volume, packet size, and timing	Medium	Enhances flexibility and adaptability for different training needs.
6	Integration with Lab Infrastructure	Ensure seamless integration with existing network defense labs and cyber ranges.	Medium	Important for deployment and interoperability with other tools and platforms.
7	Scalability	Design the traffic generator to handle large-scale traffic generation.	Medium	Crucial for testing large network environments and high-performance devices.
8	Reporting and Logging	Provide detailed logs and reports on generated traffic and detected threats.	Medium	Essential for analysis, feedback, and continuous improvement.
9	Automation and Scripting	Enable automation of traffic generation through scripting capabilities.	Medium	Increases efficiency and allows for repeatable testing scenarios.
10	Testing and Validation	Conduct extensive testing to ensure accuracy and reliability of traffic simulation.	High	Ensures the tool's effectiveness and credibility in real-world applications.
11	Documentation and User Guides	Create comprehensive documentation and guides for users.	Medium	Facilitates learning and adoption by instructors and students.
12	Performance Optimization	Optimize performance to minimize latency and maximize throughput.	High	Critical for realistic simulation and effective training in high-speed network environments.
13	Security Features	Implement security measures to protect the traffic generator and data.	Medium	Important to prevent misuse and ensure data integrity during simulations.
14	Community Support and Updates	Establish a support system and provide regular updates and improvements.	Low	Enhances long-term usability and keeps the tool up-to-date with emerging threats and protocols.

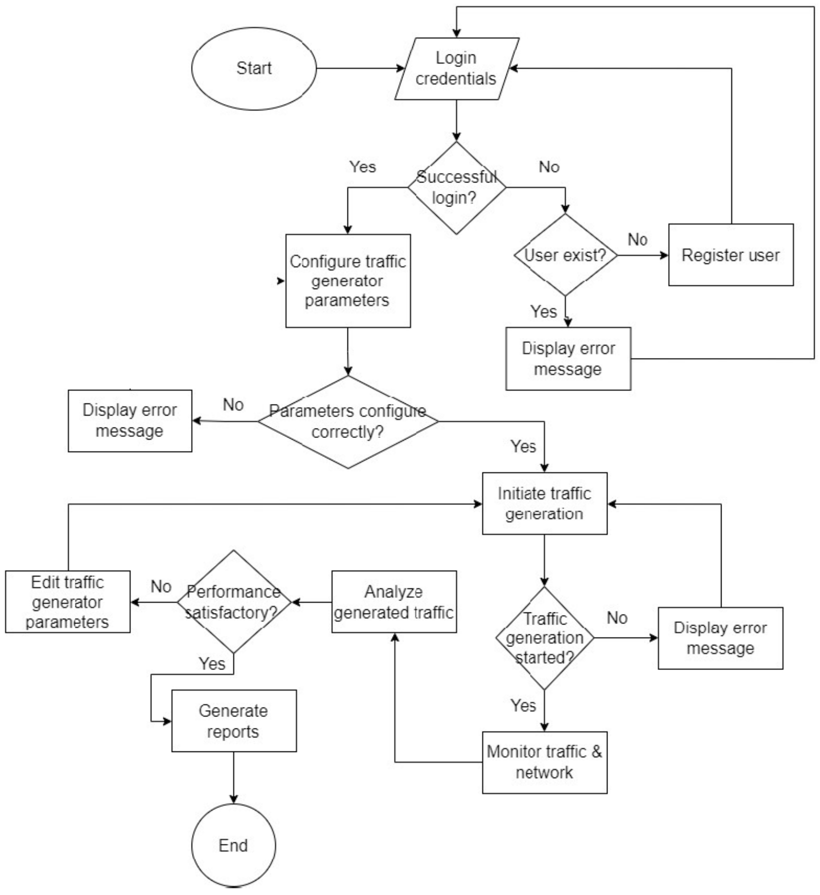

Start
Login credentials
Successful login?
Yes
No
User exist?
No
Register user
Yes
Display error message
Configure traffic generator parameters
Parameters configure correctly?
No
Display error message
Yes
Initiate traffic generation
Traffic generation started?
No
Display error message
Yes
Monitor traffic & network
Analyze generated traffic
Performance satisfactory?
No
Edit traffic generator parameters
Yes
Generate reports
End

Figure 1. Traffic Generation and Analysis.

2.1.1 Virtualized Test Environment

The experimental infrastructure was built on Proxmox Virtual Environment (VE), which provided dual virtualization capabilities through Kernel-based Virtual Machine (KVM) for hardware-assisted virtualization and Linux Containers (LXC) for lightweight OS-level virtualization. A Kali Linux virtual machine (VM) hosted on this platform served as the primary node for traffic generation and analysis. Proxmox VE's live migration feature ensured operational continuity during maintenance scenarios while optimizing resource allocation through its open-source management tools.

2.1.2 Traffic Generation Framework

Network traffic was synthesized using Cisco T-Rex in stateful mode, deployed on a dedicated physical node (PC3) within the cyber range lab. The configuration process involved three critical steps: (1) installation of T-Rex libraries and the Data Plane Development Kit (DPDK) to enable high-performance packet processing at line rate; (2) binding of network interfaces to DPDK drivers, implementing kernel bypass to eliminate OS network stack overhead; and (3) strategic allocation of Port 0 and Port 1 for bidirectional traffic transmission, with port mappings explicitly matched to the system's NICs. This optimized setup leveraged DPDK's poll-mode drivers to achieve consistent 10 Gbps throughput while maintaining sub-5% CPU utilization, ensuring efficient resource usage during extended traffic generation cycles. The architecture's design specifically accommodated both burst-type and continuous traffic patterns through T-Rex's stateful flow management capabilities.

2.1.3 Normal Traffic Synthesis

Benign traffic generation was implemented through YAML-based configuration files that defined key network parameters, including source/destination IP addresses, protocol types (HTTP, DNS, SMTP, TCP, UDP), packet transmission rates (in packets per second), and session duration. These specifications were compiled into a normalized profile (normal.yaml) and executed via T-Rex's debugging runtime (bp-sim-64-debug), which generated standardized .pcap output containing the synthetic traffic flows while maintaining protocol compliance and temporal accuracy. The resulting packet captures preserved realistic traffic characteristics, including proper protocol headers and timing distributions, suitable for baseline network behavior analysis.

2.1.4 Malicious Traffic Emulation

The SYN flood attack simulation was implemented through a structured four-phase process. First, raw attack traces were acquired and preprocessed in .pcap format to ensure data integrity. Subsequently, custom shell scripts performed flow isolation to extract single-flow packets, a necessary step for T-Rex compatibility given its traffic engine requirements. The third phase involved format conversion from .pcapng to T-Rex-compatible .pcap using specialized conversion utilities. Finally, the processed traces were integrated into a malicious.yaml profile containing attack-specific parameters including packet rates, payload sizes, and target specifications. This comprehensive methodology preserved the protocol-level authenticity of SYN flood characteristics (such as TCP flag manipulation and source IP spoofing) while maintaining full compatibility with T-Rex's stateful traffic generation capabilities, enabling high-fidelity attack simulation at line rates up to 10 Gbps.

2.1.5 Traffic Capture and Analysis

Wireshark was installed on the Kali Linux VM to analyze network traffic. The tool performed packet capture and inspection through several functions: intercepting traffic in real-time on selected interfaces, measuring throughput using I/O Graphs, analyzing protocol distributions with built-in statistics tools, and examining flow patterns to detect anomalies. This setup enabled quantitative evaluation of both normal and malicious traffic characteristics.

2.2 System Testing

Unit Testing which is validating individual components or scripts by isolation to ensure they work as intended. In this project, unit testing involved verifying YAML configuration files and scripts used for traffic generation and packet processing, DPDK port communication testing, and end-to-end flow validation.

1) Each individual YAML configuration is tested to ensure they contain valid traffic parameters such as correct IP addresses, packet rates, and durations. By running the YAML file on the TRex CLI, the configuration can be validated if it follows the correct syntax and if it contains the correct configurations. If the traffic generator successfully runs and generates traffic packets according to the configuration in the YAML file, the YAML file is configured correctly.

2) Validate shell scripts for splitting multi-flow packets into single flows or converting file formats (pcapng to pcap). With that, each flow will have one source address sending packets to one destination address. Two same configuration files with different format which is one in pcapng and another in pcap format are run on the traffic generator and the processes and output are analysed and compared. The traffic generator will run successfully because it can only handle a single flow packet.

3) The DPDK port configurations are tested to ensure NICs are bound correctly and interfaces are functional. The DPDK port needs to be properly bound with its driver in order to handle high performance traffic. Different combinations of DPDK ports are tested to ensure the traffic generator can run and generate traffic.

Another type of system testing is the Integration Testing which ensures that different components of the system such as the YAML files, traffic generator, and packet analyzer Wireshark work together as expected. The traffic generation workflow was thoroughly tested by creating and running YAML files on T-Rex and then capturing the generated traffic using Wireshark. This confirmed that the packet types, traffic rates, source and destination IP addresses, and other parameters were followed by the generated packets. By transmitting packets from one port and confirming their receipt on another port, the DPDK port configuration was also examined to also validate that it is binded properly. Furthermore, the system's entire flow was examined, beginning with the processing of .pcap files, creating traffic from the processed files, and then capturing and analyzing the traffic in Wireshark. These tests verified that the parts worked together properly to provide the intended functionality.

Furthermore, system testing focused on validating the entire system under real- world conditions to ensure that all components worked together as a cohesive unit. In order to verify that the packets fit the standards, normal traffic testing entailed creating traffic with predetermined parameters in a YAML file and examining it in Wireshark. SYN flood packets were created using processed .pcap files as part of malicious traffic testing, and Wireshark was used to confirm the malicious packet's properties. Performance testing was carried out by increasing the packets-per-second (PPS) rate and duration time in the YAML file and keeping an eye on the system's stability and resource consumption. In order to make sure the system handled problems gracefully and produced clear error messages, incorrect configurations were also introduced into error-handling scenarios. Through these tests, the system's stability, dependability, and ability to function in a variety of scenarios were confirmed.

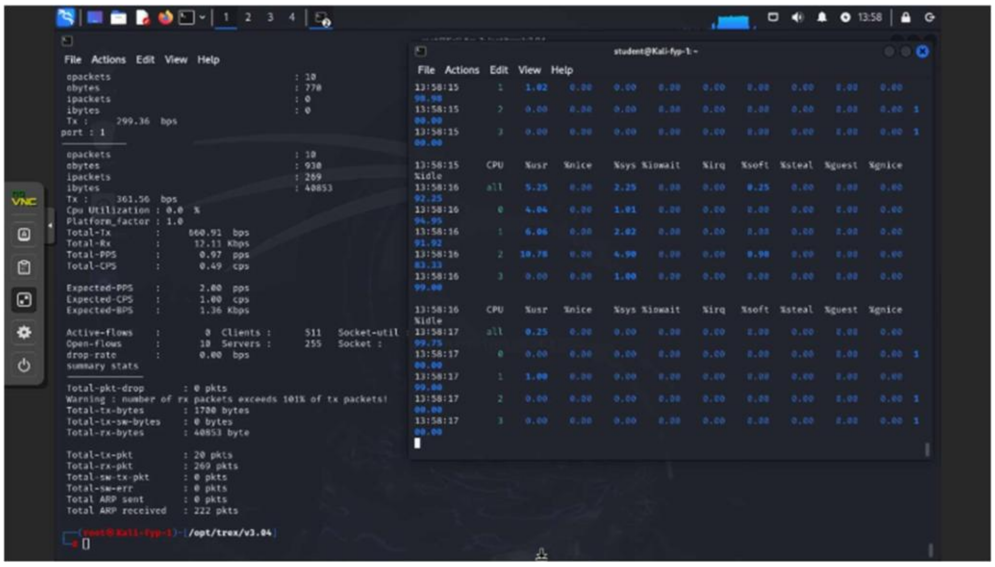
3. RESULTS AND DISCUSSION

3.1 Prototype or Wireframes

The prototyping phase involved visualizing both the traffic generated and its subsequent analysis, emphasizing the system's ability to handle diverse network scenarios effectively. A series of figures show the outcomes of normal and malicious traffic simulations, providing a comprehensive view of the tool's functionality.

3.1.1 Traffic Generated

This representation showcases the results of generating traffic based on predefined parameters. Figure 2 highlights key metrics, including packet rates, source and destination addresses, and traffic duration. It confirms the system's capability to simulate network traffic with precision, fulfilling the core objective of creating realistic traffic scenarios for training and testing purposes.



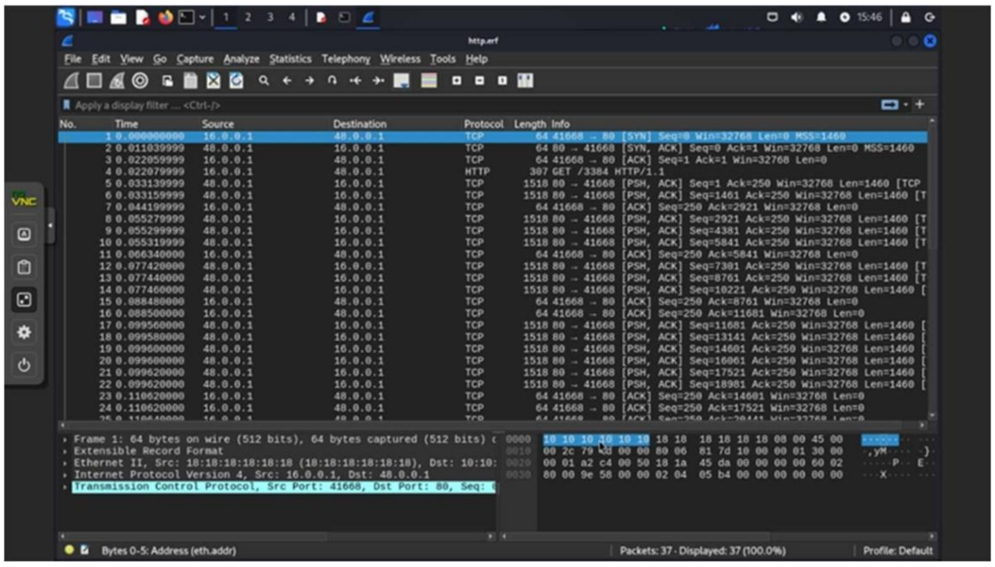
```
opackets                                    : 10
obytes                                      : 770
ipackets                                    : 0
ibytes                                      : 0
Tx :      299.36  bps
port : 1
opackets                                    : 10
obytes                                      : 930
ipackets                                    : 269
ibytes                                      : 40853
Tx :      361.56  bps
Cpu Utilization : 0.0  %
Platform_factor : 1.0
Total-Tx         :      860.91  bps
Total-Rx         :       12.11 Kbps
Total-PPS        :        0.97  pps
Total-CPS        :        0.49  cps

Expected-PPS     :        2.00  pps
Expected-CPS     :        1.00  cps
Expected-BPS     :        1.36 Kbps

Active-flows     :          0  Clients :      511   Socket-util
Open-flows       :         10  Servers :      255   Socket :
drop-rate        :        0.00  bps
summary stats

Total-pkt-drop        : 0 pkts
Warning : number of rx packets exceeds 101% of tx packets!
Total-tx-bytes        : 1700 bytes
Total-tx-sw-bytes     : 0 bytes
Total-rx-bytes        : 40853 byte

Total-tx-pkt          : 20 pkts
Total-rx-pkt          : 269 pkts
Total-sw-tx-pkt       : 0 pkts
Total-sw-err          : 0 pkts
Total ARP sent        : 0 pkts
Total ARP received    : 222 pkts

/opt/trex/v3.04
```

```
student@Kali-fyp-1:~
13:58:15      1    1.02   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00
98.98
13:58:15      2    0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00 1
00.00
13:58:15      3    0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00 1
00.00

13:58:15    CPU    %usr  %nice   %sys %iowait  %irq  %soft %steal %guest %gnice
%idle
13:58:16    all    5.25   0.00   2.25   0.00   0.00   0.25   0.00   0.00   0.00
92.25
13:58:16      0    4.04   0.00   1.01   0.00   0.00   0.00   0.00   0.00   0.00
94.95
13:58:16      1    6.06   0.00   2.02   0.00   0.00   0.00   0.00   0.00   0.00
91.92
13:58:16      2   10.78   0.00   4.90   0.00   0.00   0.98   0.00   0.00   0.00
83.33
13:58:16      3    0.00   0.00   1.00   0.00   0.00   0.00   0.00   0.00   0.00
99.00

13:58:16    CPU    %usr  %nice   %sys %iowait  %irq  %soft %steal %guest %gnice
%idle
13:58:17    all    0.25   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00
99.75
13:58:17      0    0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00 1
00.00
13:58:17      1    1.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00
99.00
13:58:17      2    0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00 1
00.00
13:58:17      3    0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00 1
00.00
```

Figure 2. Generated Traffic.

3.1.2 *Traffic Captured by Wireshark*

The traffic captured by Wireshark demonstrates the successful integration of the traffic generator with the packet analysis tool. Figure 3 displays detailed packet data, including protocol types, packet sizes, and timestamps. This integration ensures that the generated traffic aligns with the defined configurations, providing a robust environment for analyzing network behavior.

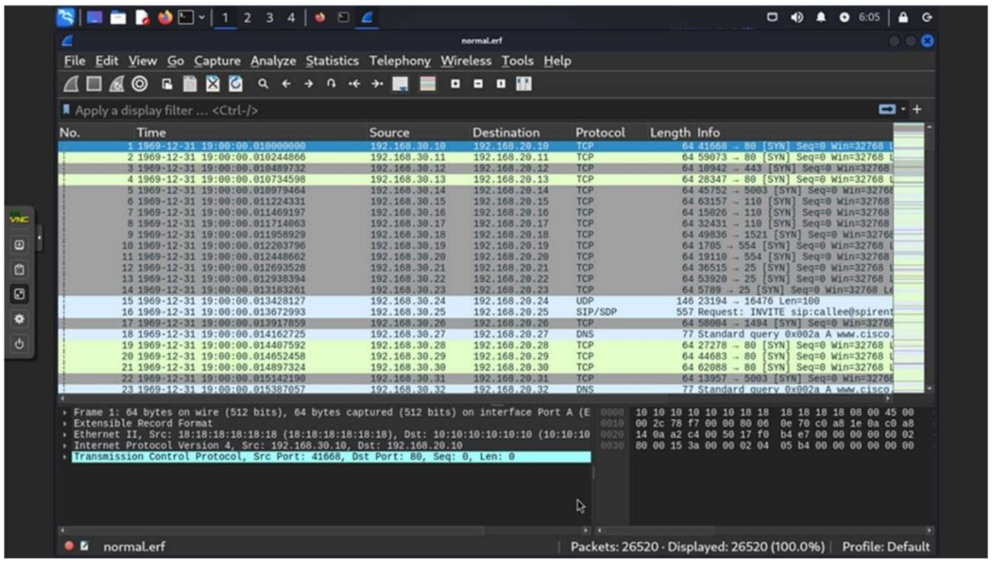


```
http.erf
No.  Time         Source     Destination  Protocol Length Info
   1 0.000000000  16.0.0.1   48.0.0.1     TCP        64 41668 → 80 [SYN] Seq=0 Win=32768 Len=0 MSS=1460
   2 0.011039999  48.0.0.1   16.0.0.1     TCP        64 80 → 41668 [SYN, ACK] Seq=0 Ack=1 Win=32768 Len=0 MSS=1460
   3 0.022059999  16.0.0.1   48.0.0.1     TCP        64 41668 → 80 [ACK] Seq=1 Ack=1 Win=32768 Len=0
   4 0.022079999  16.0.0.1   48.0.0.1     HTTP      307 GET /3384 HTTP/1.1
   5 0.033139999  48.0.0.1   16.0.0.1     TCP      1518 80 → 41668 [PSH, ACK] Seq=1 Ack=250 Win=32768 Len=1460 [TCP
   6 0.033159999  48.0.0.1   16.0.0.1     TCP      1518 80 → 41668 [PSH, ACK] Seq=1461 Ack=250 Win=32768 Len=1460 [T
   7 0.044199999  16.0.0.1   48.0.0.1     TCP        64 41668 → 80 [ACK] Seq=250 Ack=2921 Win=32768 Len=0
   8 0.055279999  48.0.0.1   16.0.0.1     TCP      1518 80 → 41668 [PSH, ACK] Seq=2921 Ack=250 Win=32768 Len=1460 [T
   9 0.055299999  48.0.0.1   16.0.0.1     TCP      1518 80 → 41668 [PSH, ACK] Seq=4381 Ack=250 Win=32768 Len=1460 [T
  10 0.055319999  48.0.0.1   16.0.0.1     TCP      1518 80 → 41668 [PSH, ACK] Seq=5841 Ack=250 Win=32768 Len=1460 [T
  11 0.066340000  16.0.0.1   48.0.0.1     TCP        64 41668 → 80 [ACK] Seq=250 Ack=5841 Win=32768 Len=0
  12 0.077420000  48.0.0.1   16.0.0.1     TCP      1518 80 → 41668 [PSH, ACK] Seq=7301 Ack=250 Win=32768 Len=1460 [T
  13 0.077440000  48.0.0.1   16.0.0.1     TCP      1518 80 → 41668 [PSH, ACK] Seq=8761 Ack=250 Win=32768 Len=1460 [T
  14 0.077460000  48.0.0.1   16.0.0.1     TCP      1518 80 → 41668 [PSH, ACK] Seq=10221 Ack=250 Win=32768 Len=1460 [
  15 0.088480000  16.0.0.1   48.0.0.1     TCP        64 41668 → 80 [ACK] Seq=250 Ack=8761 Win=32768 Len=0
  16 0.088500000  16.0.0.1   48.0.0.1     TCP        64 41668 → 80 [ACK] Seq=250 Ack=11681 Win=32768 Len=0
  17 0.099560000  48.0.0.1   16.0.0.1     TCP      1518 80 → 41668 [PSH, ACK] Seq=11681 Ack=250 Win=32768 Len=1460 [
  18 0.099580000  48.0.0.1   16.0.0.1     TCP      1518 80 → 41668 [PSH, ACK] Seq=13141 Ack=250 Win=32768 Len=1460 [
  19 0.099600000  48.0.0.1   16.0.0.1     TCP      1518 80 → 41668 [PSH, ACK] Seq=14601 Ack=250 Win=32768 Len=1460 [
  20 0.099600000  48.0.0.1   16.0.0.1     TCP      1518 80 → 41668 [PSH, ACK] Seq=16061 Ack=250 Win=32768 Len=1460 [
  21 0.099620000  48.0.0.1   16.0.0.1     TCP      1518 80 → 41668 [PSH, ACK] Seq=17521 Ack=250 Win=32768 Len=1460 [
  22 0.099620000  48.0.0.1   16.0.0.1     TCP      1518 80 → 41668 [PSH, ACK] Seq=18981 Ack=250 Win=32768 Len=1460 [
  23 0.110620000  16.0.0.1   48.0.0.1     TCP        64 41668 → 80 [ACK] Seq=250 Ack=14601 Win=32768 Len=0
  24 0.110620000  16.0.0.1   48.0.0.1     TCP        64 41668 → 80 [ACK] Seq=250 Ack=17521 Win=32768 Len=0

› Frame 1: 64 bytes on wire (512 bits), 64 bytes captured (512 bits)
› Extensible Record Format
› Ethernet II, Src: 18:18:18:18:18:18 (18:18:18:18:18:18), Dst: 10:10:
› Internet Protocol Version 4, Src: 16.0.0.1, Dst: 48.0.0.1
› Transmission Control Protocol, Src Port: 41668, Dst Port: 80, Seq:

0000  10 10 10 10 10 10 18 18  18 18 18 18 08 00 45 00
0010  00 2c 79 4d 00 00 80 06  81 7d 10 00 00 01 30 00   ,yM     }
0020  00 01 a2 c4 00 50 18 1a  45 da 00 00 00 00 60 02       P  E
0030  80 00 9e 58 00 00 02 04  05 b4 00 00 00 00 00 00     X

Bytes 0-5: Address (eth.addr)          Packets: 37 · Displayed: 37 (100.0%)          Profile: Default
```

Figure 3. Traffic captured by Wireshark.

3.1.3 *Wireshark Analysis of Normal Traffic*

A closer examination of normal traffic patterns as captured by Wireshark in figure 4. It shows the flow of typical network communications, including common protocols such as TCP, UDP, and HTTP. The figure confirms the traffic's adherence to expected parameters, offering insights into its suitability for replicating real-world scenarios in cybersecurity training environments.

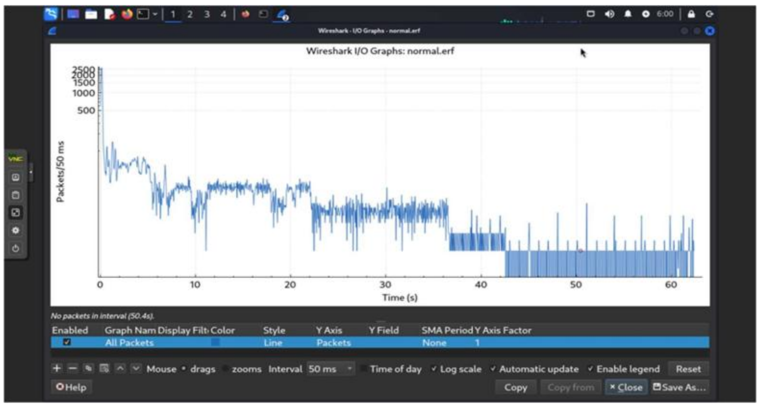

normal.erf
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
No.  Time                           Source          Destination     Protocol  Length Info
   1 1969-12-31 19:00:00.010000000  192.168.30.10   192.168.20.10   TCP       64 41668 → 80 [SYN] Seq=0 Win=32768 L
   2 1969-12-31 19:00:00.010244866  192.168.30.11   192.168.20.11   TCP       64 59073 → 80 [SYN] Seq=0 Win=32768 L
   3 1969-12-31 19:00:00.010489732  192.168.30.12   192.168.20.12   TCP       64 10942 → 443 [SYN] Seq=0 Win=32768
   4 1969-12-31 19:00:00.010734598  192.168.30.13   192.168.20.13   TCP       64 28347 → 80 [SYN] Seq=0 Win=32768 L
   5 1969-12-31 19:00:00.010979464  192.168.30.14   192.168.20.14   TCP       64 45752 → 5003 [SYN] Seq=0 Win=32768
   6 1969-12-31 19:00:00.011224331  192.168.30.15   192.168.20.15   TCP       64 63157 → 110 [SYN] Seq=0 Win=32768
   7 1969-12-31 19:00:00.011469197  192.168.30.16   192.168.20.16   TCP       64 15026 → 110 [SYN] Seq=0 Win=32768
   8 1969-12-31 19:00:00.011714063  192.168.30.17   192.168.20.17   TCP       64 32431 → 110 [SYN] Seq=0 Win=32768
   9 1969-12-31 19:00:00.011958929  192.168.30.18   192.168.20.18   TCP       64 49836 → 1521 [SYN] Seq=0 Win=32768
  10 1969-12-31 19:00:00.012203796  192.168.30.19   192.168.20.19   TCP       64 1705 → 554 [SYN] Seq=0 Win=32768 L
  11 1969-12-31 19:00:00.012448662  192.168.30.20   192.168.20.20   TCP       64 19110 → 554 [SYN] Seq=0 Win=32768
  12 1969-12-31 19:00:00.012693528  192.168.30.21   192.168.20.21   TCP       64 36515 → 25 [SYN] Seq=0 Win=32768 L
  13 1969-12-31 19:00:00.012938394  192.168.30.22   192.168.20.22   TCP       64 53920 → 25 [SYN] Seq=0 Win=32768 L
  14 1969-12-31 19:00:00.013183261  192.168.30.23   192.168.20.23   TCP       64 5789 → 25 [SYN] Seq=0 Win=32768 Le
  15 1969-12-31 19:00:00.013428127  192.168.30.24   192.168.20.24   UDP      146 23194 → 16476 Len=100
  16 1969-12-31 19:00:00.013672993  192.168.30.25   192.168.20.25   SIP/SDP  557 Request: INVITE sip:callee@spirent
  17 1969-12-31 19:00:00.013917859  192.168.30.26   192.168.20.26   TCP       64 58004 → 1494 [SYN] Seq=0 Win=32768
  18 1969-12-31 19:00:00.014162725  192.168.30.27   192.168.20.27   DNS       77 Standard query 0x002a A www.cisco.
  19 1969-12-31 19:00:00.014407592  192.168.30.28   192.168.20.28   TCP       64 27278 → 80 [SYN] Seq=0 Win=32768 L
  20 1969-12-31 19:00:00.014652458  192.168.30.29   192.168.20.29   TCP       64 44683 → 80 [SYN] Seq=0 Win=32768 L
  21 1969-12-31 19:00:00.014897324  192.168.30.30   192.168.20.30   TCP       64 62088 → 80 [SYN] Seq=0 Win=32768 L
  22 1969-12-31 19:00:00.015142190  192.168.30.31   192.168.20.31   TCP       64 13957 → 5003 [SYN] Seq=0 Win=32768
  23 1969-12-31 19:00:00.015387057  192.168.30.32   192.168.20.32   DNS       77 Standard query 0x002a A www.cisco.

› Frame 1: 64 bytes on wire (512 bits), 64 bytes captured (512 bits) on interface Port A (E
› Extensible Record Format
› Ethernet II, Src: 18:18:18:18:18:18 (18:18:18:18:18:18), Dst: 10:10:10:10:10:10 (10:10:10
› Internet Protocol Version 4, Src: 192.168.30.10, Dst: 192.168.20.10
› Transmission Control Protocol, Src Port: 41668, Dst Port: 80, Seq: 0, Len: 0

0000  10 10 10 10 10 10 18 18  18 18 18 18 08 00 45 00
0010  00 2c 78 f7 00 00 80 06  0e 70 c0 a8 1e 0a c0 a8
0020  14 0a a2 c4 00 50 17 f0  b4 e7 00 00 00 00 60 02
0030  80 00 15 3a 00 00 02 04  05 b4 00 00 00 00 00 00

normal.erf    Packets: 26520 · Displayed: 26520 (100.0%)    Profile: Default
```

Figure 4. Wireshark representation of Normal Traffic.

3.1.4 Graphical Visualization of Normal Traffic

Figure 5 represents a visual analysis of normal traffic over time. It highlights consistent trends in packet rates and protocol distribution, providing a baseline for understanding standard network behavior. Such visualizations enhance the interpretation of traffic patterns, making them a valuable tool for identifying anomalies during training exercises.



```
No packets in interval (50.4s).
Enabled  Graph Nam  Display Filt  Color  Style  Y Axis   Y Field  SMA Period  Y Axis Factor
  ✓      All Packets                     Line   Packets           None        1

Mouse • drags  zooms   Interval 50 ms   Time of day  ✓ Log scale  ✓ Automatic update  ✓ Enable legend  Reset
Help                                           Copy   Copy from   × Close   Save As…
```

Figure 5. Visualization for Normal Traffic

3.2 Malicious Traffic Characterization

The testing framework successfully identified and analyzed six distinctive characteristics of malicious SYN flood traffic, as detailed in Table 4.

Table 4 Features of Malicious Traffic

Feature	Explanation
Abnormal Packet Rate	The traffic exhibits an extremely high packet rate of 400,000 packets per second (PPS), which far exceeds typical network activity and is unsustainable for most systems.
Short Time Span	A substantial volume of packets (39,999) is transmitted within a very short duration (0.1 seconds), leaving insufficient processing time for legitimate traffic.
Small Packet Sizes	The packets are deliberately small (64 bytes), allowing for the rapid transmission of high-volume traffic to saturate the network's bandwidth and resources.
Repetitive SYN Packets	The traffic predominantly consists of SYN packets without completing the TCP handshake, which disrupts the server's normal operation and consumes resources.
Unusual Graph Pattern	The graphical analysis reveals a sharp, linear spike in packet transmission, which is atypical and indicative of automated malicious activity.
Unfinished TCP Handshakes	The SYN packets are sent without corresponding ACK responses, intentionally creating half-open connections and exhausting server resources.

These features were systematically evaluated to validate the detection system's effectiveness:

a. Abnormal Packet Rate: The packet rate reached 400,000 PPS, significantly exceeding typical thresholds and overwhelming the network capacity.
b. Short Time Span: Malicious packets were transmitted rapidly within a span of 0.1 seconds, leaving no room for legitimate traffic to be processed.
c. Small Packet Sizes: The smaller packet size of 93 bytes facilitated a high transmission rate, saturating bandwidth and exhausting system resources.
d. Combination of Protocols: The simultaneous use of TCP and UDP increased the attack complexity and strain on the system's defense mechanisms.
e. Unusual Graph Pattern: The linear spike in traffic volume is a hallmark of automated attacks, such as DoS or SYN Flood, where traffic increases abruptly without fluctuation.
f. Unfinished TCP Handshakes: SYN packets were sent without ACK responses, leaving half-open connections that drained server resources, disrupting normal operations.

3.3 Traffic Generation and Attack Simulation Validation

Test case tested for this project includes:

1) Verifying Normal Traffic Generation

Table 5 Verify Normal Traffic Test Case 1 Table

Objective	To verify that the T-Rex traffic generator produces normal traffic with the specified parameters
Steps	1. Create a YAML file with predefined configuration for normal traffic parameters. 2. Execute the traffic generator with the command: 'sudo ./bp-sim- 64-debug -f normal.yaml -o normal.erf'. 3. Capture the traffic using Wireshark. 4. Analyze the packet captured with the predefined parameters.
Expected	Wireshark should display packets with the specified source and destination IPs,

Outcome	packet types, and packet rates.
Actual Outcome	Wireshark showed multiple types of packets with the correct source and destination IPs, and packet rates.
Pass/Fail	Pass

2) Verifying Malicious Traffic Generation

Table 6 Verify Malicious Traffic Test Case 2 Table

Objective	To verify that the T-Rex traffic generator can create malicious SYN flood traffic based on a processed .pcap file.
Steps	1. Process a multi-flow SYN flood .pcap file to split it into single- flow packets using a shell script. 2. Convert the resulting .pcap file to .erf format using T-Rex. 3. Create a YAML file named malicious.yaml referencing the .erf file. 4. Execute the traffic generator with the command: sudo ./bp-sim-64-debug -f malicious.yaml -o malicious.erf. 5. Capture and analyze the traffic in Wireshark on the destination interface.
Expected Outcome	Wireshark should display SYN packets with a high frequency targeting a single destination, characteristic of SYN flood traffic.
Actual Outcome	Wireshark showed a high volume of SYN packets targeting the specified destination, consistent with SYN flood attack behavior.
Pass/Fail	Pass

Table 7 provides a comparative analysis of captured traffic parameters for normal and malicious scenarios, highlighting key distinctions in packet behavior and threat characteristics.

Table 7 Normal and Malicious Traffic Comparison

Features	Normal Traffic	Malicious Traffic
Captured Packets	26,520	438,999
Time Span	62.378 seconds	0.1 seconds
Average PPS	425.1	439,000
Average Packet Size	651 bytes	93 bytes
Protocols	TCP, UDP, DNS, SMTP, HTTP, POP	TCP, UDP
Graph Characteristics	Fluctuating packet rate	Linear spike in packet volume
Observed Behavior	Typical network Communications	Abnormal traffic, indicative of DoS
Potential Threat Level	Low	Critical

The comparison of normal and malicious traffic demonstrates significant differences in network behavior. Normal traffic, captured over 62.378 seconds, exhibited an average packet rate of 425.1 packets per second (PPS) with a consistent fluctuation in packet volumes. The average packet size was 651 bytes, and the protocols used included TCP, UDP, DNS, SMTP, and HTTP, all indicative of regular network communication. This traffic is categorized as having a low threat level, as no suspicious patterns were observed in its characteristics or graph representation.

Conversely, malicious traffic, generated as part of SYN flood and DoS attack scenarios, exhibited critical anomalies. A total of 438,999 packets were transmitted within a brief span of 0.1 seconds, resulting in an overwhelming packet rate of 439,000 PPS. The packets, with an average size of 93 bytes, were intentionally small to maximize volume and disrupt system operations. Malicious traffic primarily utilized TCP and UDP protocols, and its graph displayed a sharp, linear spike in packet volumes, strongly indicating automated attack behavior.

3) Verifying Packet Capture with Wireshark

Table 8 Verify PCAP with Wireshark Test Case 3 Table

Objective	To confirm that Wireshark accurately captures and displays traffic generated by T-Rex.
Steps	1. Generate normal traffic using a preconfigured YAML file (normal.yaml). 2. Start Wireshark on the receiving interface and begin capturing traffic. 3. Let the traffic generation run for 10 seconds. 4. Stop the Wireshark capture and analyze the captured packets.
Expected Outcome	Wireshark should capture packets matching the specified parameters.
Actual Outcome	Wireshark captured and displayed packets matching the expected parameters.
Pass/Fail	Pass

4. CONCLUSION AND FUTURE WORK

The development of the traffic generator was driven by the need for customizable, scalable, and cost-effective solutions for cybersecurity training. The project utilized tools like Cisco TRex and Wireshark to simulate and analyze network traffic. Key milestones included designing the architecture, implementing the traffic generation engine, and validating results through rigorous testing. Significant achievements included creating YAML-based configurations for both normal and malicious traffic, as well as enabling real-time packet analysis. achieved the goals by developing a versatile traffic generator that supports various protocols (e.g., TCP, UDP, HTTP) and simulates attack scenarios like SYN floods. The system integrates seamlessly with existing network labs and supports real-time analysis. YAML syntax flaws, DPDK port setup problems, and the conversion of multi-flow PCAP files into single flows for malicious traffic simulation have all been fixed. With dependable packet creation, comprehensive traffic analysis, and user-friendly interfaces, the main goals were entirely met. While the recommendation for future works includes adding support for more attack scenarios, such as SQL injection and ransomware, introduce random attack generation to simulate real-world cyber threats dynamically, optimize server performance to handle larger-scale traffic generation, develop a web-based GUI for easier configuration and traffic monitoring, and integrate machine learning to analyze traffic patterns for enhanced threat detection

REFERENCES

[1] Katsantonis, M. N., Manikas, A., Mavridis, I., Gritzalis, D., 2023. Int. J. Inf. Secur. 22(4), 1005–1027.

[2] Bistene, J. V., das Chagas, C. E., dos Santos, A. F. P., Salles, R. M., 2024. Modeling Network Traffic Generators for Cyber Ranges: A Systematic Literature Review.

[3] Grimaldi, A., Ribiollet, J., Nespoli, P., Garcia-Alfaro, J., 2024. Toward Next-Generation Cyber Range: A Comparative Study of Training Platforms, 271–290.

[4] Beuran, R., 2025. Cybersecurity Education and Training, in: Cybersecurity Education and Training, 9–18.

[5] Kokkonen, T., Hämäläinen, T., Silokunnas, M., Siltanen, J., Zolotukhin, M., Neijonen, M., 2015. Analysis of Approaches to Internet Traffic Generation for Cyber Security Research and Exercise, 254–267.

[6] Bistene, J. V., das Chagas, C. E., dos Santos, A. F. P., Salles, R.M., 2024. Modeling Network Traffic Generators for Cyber Ranges: A Systematic Literature Review.

[7] Adeleke, O. A., Bastin, N., Gurkan, D., 2023. ACM Comput. Surv. 55(2), 1–23.

[8] Angrisani, L., Botta, A., Miele, G., Vadursi, M., 2013. IEEE Int. Workshop on Measurements & Networking, 74–78.

[9] Cisco, 2023. TRex.

[10] Swann, M., Rose, J., Bendiab, G., Shiaeles, S., Savage, N., 2020. World Congress on Internet Security, 24–29.

[11] Ghosh, S. K., Satvat, K., Gjomemo, R., Venkatakrishnan, V. N., 2022. Ostinato: Cross-host Attack Correlation Through Attack Activity Similarity Detection, 1–22.

[12] Patil, B. R., Moharir, M., Mohanty, P. K., S. G., S. S., 2017. Int. Conf. on Computational Systems and Information Technology for Sustainable Solution, 1–5.

[13] Swann, M., Rose, J., Bendiab, G., Shiaeles, S., Savage, N., 2020. World Congress on Internet Security, 24–29.

[14] Molnar, S., Megyesi, P., Szabo, G., 2013. IEEE Int. Conf. on Communications Workshops, 1340–1344.