**PacketSleuth: Comprehensive Simulation, Defense, and Detection of MITM Attacks**

**Abstract** PacketSleuth is a cybersecurity project focused on simulating Man-in-the-Middle (MITM) attacks, implementing effective defense mechanisms, and utilizing machine learning to detect such attacks automatically. By capturing normal and attack network traffic, preprocessing it into structured datasets, and training a TinyBERT model, this project demonstrates a full lifecycle security approach. Key challenges included traffic parsing and model overfitting, which were addressed through script refinements and extended data collection. The project ultimately validated the hypothesis that MITM conditions can be detected using encrypted traffic's structural characteristics.

**Introduction** Man-in-the-Middle attacks pose significant threats to network communications, allowing attackers to intercept, alter, or forge data exchanges between devices. Detecting these attacks before significant damage occurs is crucial. The PacketSleuth project aimed to simulate realistic MITM attacks, implement common defensive mechanisms, and train a lightweight transformer-based machine learning model to detect attacks using observable packet features without relying on payload contents. This paper documents the project's technical design, development process, challenges, and results.

**Background** The technical stack leveraged industry-standard tools and frameworks. Ettercap was used for ARP poisoning-based MITM simulation, Wireshark's tshark for packet capture, and Scapy for parsing captured traffic. Flask served as the platform for building an HTTPS server, protected by OpenSSL certificates and HTTP Strict Transport Security (HSTS) headers. Python's pyOpenSSL library enabled SSL pinning in the client. Machine learning components relied on HuggingFace's TinyBERT model, fine-tuned using PyTorch and the HuggingFace Datasets library. Supporting utilities included Brew for package management, Wireshark for manual inspection, and a Makefile to automate capture and attack commands.

**Methodology**

**Traffic Capture and Attack Simulation:** A Python script (record.py) was created to capture traffic using tshark, targeting a specific network interface. It accepted configurable arguments such as duration and filename prefix. Two types of captures were conducted: normal network activity and under active MITM attack using Ettercap. Ettercap performed ARP poisoning on the victim and the gateway, redirecting all traffic through the attacker's device.

**Defense Implementation:** A Flask-based HTTPS server (server.py) was built, secured with OpenSSL-generated self-signed certificates. HTTP Strict Transport Security (HSTS) headers were enforced. To defend clients, an SSL pinning script (pinning\_client.py) was implemented. It connected to the server, extracted the SSL certificate fingerprint, and compared it to the known good fingerprint to detect MITM attempts.

**Data Preprocessing and Dataset Creation:** Captured PCAP files were parsed using a custom script (build\_dataset.py) leveraging Scapy. Early parsing efforts failed to extract full packet information. The parsing logic was corrected to focus on IP packets and specifically TCP and UDP layers. Extracted fields included source IP, destination IP, protocol type, and packet length. These fields were serialized in JSON Lines (JSONL) format. Packets were labeled "normal" or "attack" based on the session during which they were captured.

**Machine Learning Model Fine-tuning:** The dataset was split into 90% training and 10% evaluation sets. HuggingFace Transformers library and TinyBERT were utilized for fine-tuning. The model was trained on the structured packet representations for three epochs, using PyTorch as the backend. The training script (train.py) was enhanced to save example packet inputs for live testing with the live\_detect.py script, which takes user inputs and predicts whether the packet is normal or under attack.

**Experiment Details:**

* Number of packets captured: ~5,000
* After filtering and labeling: ~2,800 packets for training
* Attack traffic proportion: 40% attack, 60% normal
* Average training time per epoch: ~1 minute

**Visualization of Workflow:**

[Packet Capture]

↓

[Attack / Normal Traffic Separation]

↓

[Scapy Parsing and Feature Extraction]

↓

[JSONL Dataset Creation]

↓

[TinyBERT Fine-tuning]

↓

[Live Detection via Input Prompts]

**Results** PacketSleuth demonstrated that encrypted traffic's structural features provide sufficient information for detecting MITM attack scenarios. Despite payload encryption, differences in packet lengths, source-destination patterns, and protocol distributions allowed the TinyBERT model to achieve robust classification performance. Extending capture sessions significantly improved dataset diversity, directly improving model generalization. Live detection experiments successfully predicted packet scenarios based on manual input, reinforcing model viability.

**Technical Stack Details:**

* **Capture/Parsing:** Wireshark tshark, Scapy
* **Attack Simulation:** Ettercap
* **Defense:** Flask (HTTPS server), OpenSSL (SSL certs), pyOpenSSL (pinning client)
* **Machine Learning:** HuggingFace Transformers, TinyBERT, Datasets library, PyTorch
* **Automation/Utilities:** Makefile, Brew package management, Wireshark manual analysis

**Challenges and Solutions:**

* **Initial Parsing Failure:** Resolved by refining Scapy layer selection and filtering logic.
* **Dataset Imbalance:** Extended capture durations to balance attack/normal traffic.
* **Overfitting on Small Dataset:** Mitigated by increasing dataset size and diversity.
* **Live Detection Prediction Format:** Enhanced post-training logging to validate expected input-output formats.

**Conclusion** PacketSleuth successfully delivered a complete cybersecurity demonstration of the MITM attack lifecycle, from simulation to defense to detection. It effectively overcame development challenges and validated structural pattern analysis as a viable means for detecting MITM conditions, even on encrypted traffic. Future work could focus on incorporating packet timing features, expanding dataset variety, and creating real-time detection pipelines for deployment in operational environments.

Nick Deveau - 2025