Atividade 3 e 4: Setup experimental de rede Ethernet e Sistema de Deteccão de Intrusão

Gabriel Soares[1], **Maria Eduarda Melo**[1]

[1]Centro de Informática – Universidade Federal de Pernambuco (UFPE)
Recife – PE – Brasil

{gss12, meom}@cin.ufpe.br

Resumo. *Este relatório descreve o desenvolvimento de um sistema de detecção de intrusão (IDS) baseado em aprendizado de máquina para redes Ethernet automotiva. O projeto abrange a configuração de uma rede Ethernet com TSN Boxes, a criação de datasets com o tráfego normal e anormal da rede e a avaliação dos algoritmos K-Means, Isolation Forest, One Class SVM e Auto-encoder LSTM na sua capacidade para classificar esses tráfegos, chegando ao melhor resultado com o Isolation Forest.*

1. Introdução

A crescente complexidade dos sistemas automotivos modernos tem impulsionado a adoção de tecnologias avançadas de comunicação, como a Ethernet Automotiva. Essa tecnologia promete maior largura de banda e menor latência em comparação com os sistemas tradicionais, facilitando a integração de diversas funcionalidades críticas e de entretenimento nos veículos. No entanto, a crescente conectividade também traz à tona a preocupação com a segurança cibernética, uma vez que ataques e intrusões podem comprometer a integridade e a funcionalidade dos sistemas automotivos.

Este relatório aborda a problemática da segurança em redes Ethernet automotivas, focando no setup experimental de uma rede Ethernet automotiva (protocolo AVTP - IEEE 1722), desenvolvimento e implementação de um Sistema de Detecção de Intrusão (IDS) baseado em aprendizado de máquina. A motivação para este trabalho reside na necessidade de detectar e mitigar possíveis ataques em redes automotivas, garantindo a segurança dos veículos e, consequentemente, dos seus ocupantes.

2. Arquitetura proposta

A arquitetura do sistema proposto nessa atividade é um setup experimental de rede Ehetnet automotiva (protocolo AVTP - IEEE 1722) com as TSN Boxes do TSN Systems. São ao todo três Boxes (tap, talker e listener) conectadas entre si por portas e ao computador através de um switch Ethernet tradicional. Através do computador é possível acessar o software do TSN Systems para configuração e uso das Boxes, além do acesso ao sofware do Wireshahrk para análise dos pacotes transmitidos na rede.

3. Metodologia e validação experimental

3.1. Datasets

Para que fosse possível treinar e testar os algoritmos, foram criados datasets de tráfego normal, datasets para cada uma das modificações com três combinações de parâmetros e

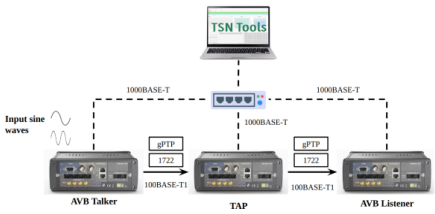

Figura 1. Arquitetura do setup experimental da rede Ethernet

um dataset com todas as modificações e com apenas uma combinação de parâmetros para cada modificação. As modificações são descriminadas mais datalhadamente abaixo:

1. **Drop:** Essa modificação faz com que alguns pacotes sejam perdidos, fazendo com que o delay do próximo pacote recebido fuja do padrão esperado, sendo maior em relação ao último pacote recebido. A combinação de parâmetros usadas para o Action Count, Action Parameter e Interval Count, respectivamente, foram: 2, -1, 10000000; 12, -1, 30000000; 8, -1, 800000.
2. **Delay:** Tal modificação segura o envio dos pacotes por determinado período de tempo, soltando todos os pacotes acumulados logo em seguida, isso altera o tempo de delay com relação ao último pacote recebido para todos os pacotes afetados. A combinação de parâmetros usadas para o Action Count, Action Parameter e Interval Count, respectivamente, foram: 7, -1, 9000000; 2, -1, 4000000; 11, -1, 14000000.
3. **Out Of Sequence:** Essa modificação troca certos pacotes de posição, o que termina afetando o delay com relação ao último pacote dos dois pacotes que foram trocados, já que é preciso esperar o segundo pacote para inverter as posições. A combinação de parâmetros usadas para o Action Count, Action Parameter e Interval Count, respectivamente, foram: 3, -1, 10000000; 1, -1, 13000000; 8, -1, 20000000.
4. **Jitter:** A modificação aplica alguns ruídos no delay de alguns pacotes, a combinação de parâmetros usadas para o Action Count, Action Parameter e Interval Count, respectivamente, foram: 80000, -1, 10000000; 600000, -1, 25000000; 50000, -1, 4000000.

3.2. Validação Experimental

Antes que os dados pudessem ser utilizados, eles passaram por uma fase simples de tratamento que envolveu a seleção das features mais importantes, no nosso caso, ficamos com a coluna de Time, que representava o delay de um pacote com relação ao anterior, e a coluna de Length, que continha o tamanho do pacote. Sendo necessário em seguida, apenas, a normalização dos dados, específicas de cada modelo.

Em seguida, foi necessário rotular o dataset, para isso, foram usadas as colunas Time e Length, á que a maioria dos pacotes tinha um delay com relação ao pacote anterior entre 0.000124 e 0.000126. Os únicos que fugiam do padrão eram as mensagens de Sync, que possuiam um Length diferente, e os pacotes que viam imediatamente depois deles. Logo, o script em Python utilizado para a rotulação considerava como tráfego modificado os pacotes com o delay foram do range esperado e que tivessem tanto o seu Length, quanto o Length do pacote anterior iguais a 66.

O conjunto de treinamento se resumiu a gravação do tráfego da rede sem que essa tivesse sido exposta a nenhuma modificação dos pacotes, esse conjunto possuiu ao total 969547 instâncias. Já para a testagem, foram usadas gravações separadas para cada um dos tipos de modificadores de pacote, os quais continham três variações da modificação e que continham os valores de 1417617 (Drop), 1449461 (Delay), 1451584 (Out Of Sequence) e 1453824 (Jitter). Com relação a testagem dos algoritmos, foi utilizada uma gravação com os quatro tipos de modificações, com apenas uma combinação de parâmetros para cada, a qual tinha 196481 instâncias.

4. Resultados e discussões

Depois de finalizada as fases de treinamento, para que os modelos pudessem aprender o comportamento da rede, sua eficácia na previsão de possíveis erros ou modificações intencionais realizados sob os pacotes foi posta à prova na fase de testagem, sendo obtidos os seuintes resultados:

Modelo	AUC	Acurácia	Precisão	F1-Score
KMeans	0.9975	0.9906	0.8875	0.9404
Isolation Forest	0.9974	0.9949	0.9227	0.9598
OCSVM	0.9950	0.9906	0.8662	0.9283
Autoencoder LSTM	0.9069	0.9694	0.7723	0.8017

Tabela 1. Resultados de desempenho dos algoritmos

Apesar de simples, o algoritmo K-Means obteve bons resultados, isso pode ser justificado pela natureza mais simples do problema, em que os dados poderiam ser separados em grupos sem muita dificuldade. O modelo de Autoencoder LSTM, apesar de ser mais bem preparado para lidar com dados em sequência, que tenham alguma ligação temporal, foi o que obteve as piores classificações, sendo necessário realizar novas testagens com diferentes combinações de parâmetros e por um número maior de épocas com um valor menos de batches para tentar encontrar resultados melhores. Os algoritmos baseados na estratégia de One Class Novelty Detection, obtiveram bons resultados, com destaque para o Isolation Forest, que chegou a alcançar o resultado de 0.9598 de f1-score.

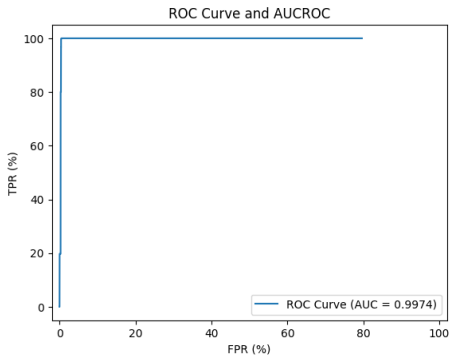

Figura 2. Gráfico da curva ROC do modelo Isolation Forest

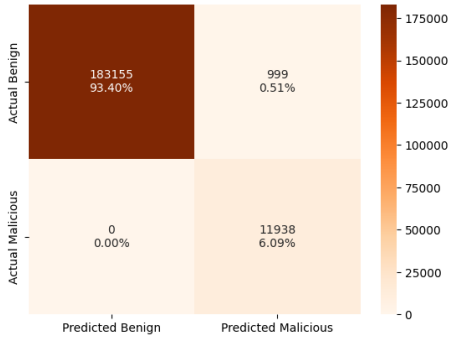

Figura 3. Matriz de confusão do modelo Isolation Forest

5. Conclusão e trabalhos futuros

Nesse trabalho, propomos um IDS não supervisionado para detectar modificações, ou possíveis erros, na transmissão de pacotes em uma rede Ethernet com protocolo IEE 1722. Nossa solução não requer o uso de dados maliciosos no treinamento, sendo o modelo capaz de aprender a detectar anomalias a partir do entendimento do comportamento normal da rede. Durante os testes, o melhor resultado de f1-score foi de 0.9598, obtido pelo algoritmo de detecção de anomalia Isolation Forest. Para trabalhos futuros, é preciso entender alternativas para realização do deploy do IDS na rede apre.

Referências

[1] TSN Box Quick Start. Disponível em: https://jira.tsn.systems/confluence/display/TSSD/TSN+Box+-+Quick+Start.

[2] Audio Streaming. Disponível em: https://jira.tsn.systems/confluence/display/TSSD/1722+Talker. Acesso em 13 de Abril de 2024.

[3] Packet Stream Modifiers. Disponível em: https://jira.tsn.systems/confluence/display/TSSD/Packet+Stream+