Análise Exploratória de Ataques por meio de Instruções em Código de Máquina em Automóveis Inteligentes

Bruno H. Labres[1], **Ovídio J. Silva J.**[1]

[1]Departamento de Informática – Universidade Federal do Paraná (UFRGS)
Curitiba - PR - Brasil

{bhl16,ojsj18}@inf.ufpr.br

Resumo. *O uso de IoT em veículos acarreta no aparecimento de novos alvos para exploradores de falhas que podem resultar em acidentes fatais. Esse trabalho realiza uma análise exploratória de ataques que usam instruções em código de máquina em automóveis inteligentes. O pipeline contará com um classificador para detectar se a instrução é maliciosa, e, caso seja, um classificador para identificar o tipo de ataque.*

1. Introdução

O uso de recursos inteligentes em carros tem aumentado com os avanços tecnológicos, este avanço tem como consequência o aumento de conectividade externas, como servidores para direção autônomas, uso de sensores geográficos e de orientação e etc. Tais funcionalidades abrem brechas para ataques de disponibilidade, que em casos como os veiculares, envolvem a segurança do usuário [Kang et al. 2021]. Esses fatores abrem espaço para uma nova área focada em segurança em IoT automobilístico.

2. Descrição do problema

Os veículos na sua maior parte se utilizam do protocolo CAN(*Controller Area Network*), um modelo de transmissão de informações através de nós entre microcontroladores e dispositivos sem a necessidade de um computador host, para que tudo ocorra de maneira segura e eficiente as mensagens são divididas no padrão da imagem a seguir: Tendo um sistema extremamente regrado, os ataques se utilizam de recursos do modelo de mensagens para quebrar a conexão entre os nós do sistema veicular. Com isso, seria útil identificar possíveis padrões que ocorrem em ataques a esses sistemas.

3. Motivação

Como é uma área relativamente recente, a escassez de pesquisa focada nesse campo se caracteriza como algo urgente, já que ataques nesse nível podem resultar diretamente em danos graves a indivíduos. Com o avanço na área de *IoT*, esperasse que essas pesquisas se tornem mais recorrentes na academia e no mercado.

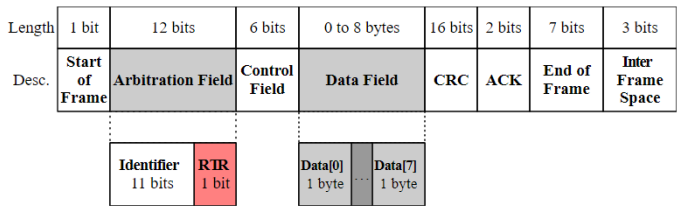

Figure 1. Encapsulamento da mensagem no padrão CAN

4. Visão geral do conjunto de dados

4.1. Coleta

Os conjuntos de dados foram coletados através de diferentes bancos disponibilizados pelo Hacking and Countermeasure Research Lab [Seo et al. 2018]. Esses dados incluem exemplos de ataques por negação de serviço, fuzzy attack, *spoofing* em RPM e *spoofing* na engrenagem motriz. Os conjuntos de dados foram obtidos através dos logs do tráfico de CAN na porta OBD-II de um veículo enquanto ataques por injeção de mensagem ocorriam.

Os dados foram coletados através de quatro arquivos CSV disponibilizados pelo laboratório, um para cada tipo de ataque.

- **Rodada Preliminar**: Nesta rodada os dados foram uma coleta de ataques realizados para análise e pesquisa, logo o desenvolvimento proposto era de maneira estática.
- **Rodada final**: Nesta rodada os dados foram gerados com o intuito de atacar os protocolos, divididos em até 5 tipos de ataques, e que fossem utilizados maneiras dinâmicas de resolver o problema.

4.2. Atributos

Os atributos de cada tabela estão organizados no seguinte esquema:

- **Timestamp**: horário em que a instrução foi registrada;
- **CAN ID**: identificador da mensagem CAN em hexadecimal;
- **DLC**: número de bytes de dados, de 0 a 8;
- **DATA[0 7]**: campos de dados (por byte);
- **Flag**: T ou R, T representa mensagem injetada enquanto R representa mensagem normal.

4.3. Pré-processamento

Foram obtidos quatro arquivos CSV, um para cada tipo de ataque e a partir destes obtivemos instruções não-maliciosas também, para compor uma quinta classe. Cada tabela foi transformada em um *dataframe* do Pandas com os campos equivalentes aos do banco. A partir disso, usamos os quatro *dataframes* disponíveis para construir um *dataframe* resultante, onde possui 10.000 amostras de ataques de cada *dataframe* e mais 10.000 amostras de mensagens que não são ataques (obtidas no dataframe de *fuzzy attack*. Com isso, temos cinco classes.

Além disso, os campos de dados foram convertidos de hexadecimal para decimal. Isso facilitará a extração de características na etapa seguinte.

4.4. Extração de características

Para o escopo do projeto, escolhemos fazer a extração de característica com base nos oito campos de dados (cada um representando um byte). Essa escolha foi feita pelo seguinte motivo: os campos de *timestamp* e *CAN ID* podem ser relevantes para identificar ataques, porém para isso seria necessário avaliar a série temporal de amostras, já que são ataques de injeção de mensagens com um certo período ou intervalo de tempo, o que não seria viável no escopo deste projeto. O campo DLC é fixado em 8 bytes. O campo de flag

foi pré-processado na escolha de classes (já que ele indica se é ou não uma mensagem injetada). Com isso, o campo utilizado para a extração de características são os oito campos de dados. A transformação para dados numéricos foi feita no pré-processamento para assim facilitar a extração de características. Seguindo as recomendações em aula, como os dados foram pré-processados para numéricos, eles são normalizados e, com isso, as características são obtidas. O vetor de características possui oito elementos e é obtido a partir dos bytes de dados das instruções CAN.

4.5. Classes

Nós optamos por transformar as mensagens que não são ataques em uma classe com o mesmo peso das outras, para assim fazer comparações através dessa perspectiva. Os exemplos resultantes pertencem a cinco classes distintas: *normal* ,*DoS*, *fuzzy attack*, *spoofing* na RPM e *spoofing* na engrenagem motriz. As classes são balanceadamente distribuídas, onde cada uma possui 10.000 amostras, totalizando 50.000 amostras para o conjunto de dados.

5. Planejamento futuro

A área tem como um de seus objetivos identificar através dos logs do sistema mensagens que tem como alvo atrapalhar o funcionamento do sistema, logo a detecção destes ataques seria o ideal para evitar que os comandos cheguem a afetar o sistema. A variedade nas formas que os ataques acontecem influencia na restrição que será utilizado para validar os dados, um exemplo é que os ataques do tipo DoS tem um grau de dificuldade de identificação menor, pois trabalham com instruções repetidas em curtos períodos de tempo, diferente do fuzzy attack que simula um log de uso comum, dificultando a identificação, a falta de equilíbrio nestes dados podem acarretar em problemas de overfitting e Underfitting, em ambos os casos o trabalho de conexões entre os sensores do carro são afetados.

Optou-se por utilizar os modelos KNN(*K-Nearest Neighbors*) e *Random Forest* que trabalham bem como a proposta do *dataset* ser numérico e rotulado. Os dados estão divididos pelos tipos de ataques, incluindo uma classe de instruções não-maliciosas. No caso do KNN, será interessante visualizar os agrupamentos por ataque de acordo com as amostras. A random forest também pode fornecer resultados satisfatórios mesmo sem o ajuste fino de seus hiperparâmetros, o que é bastante útil em uma análise exploratória. O pipeline do projeto consiste em dois classificadores: um classificador para testar se o exemplo é malicioso, e, caso seja, um classificador para diagnosticar o tipo de ataque em questão.

References

Kang, H., Kwak, B. I., Lee, Y. H., Lee, H., Lee, H., and Kim, H. K. (2021). Car hacking: Attack defense challenge 2020 dataset.

Seo, E., Song, H. M., and Kim, H. K. (2018). Gids: Gan based intrusion detection system for in-vehicle network. In *2018 16th Annual Conference on Privacy, Security and Trust (PST)*, pages 1–6.