놀라운 CAN 공격 탐지 기법

최슬기*, 박정연*, 권혁민**, 김종민**, 이상욱**
(*이화여자대학교 사이버보안전공, **고려대학교)

I. 서론

ECU를 시작으로 차량 내부의 각종 전자장비 채용이 늘어남에 따라 차량이라는 시스템은 이전의 기계적인 장치가 아닌 전자장비를 활용하는 하나의 차량 네트워크를 구성하게 되었다. 이러한 차량 네트워크 통신은 CAN 패킷을 활용하여 이루어지고, 차량 네트워크는 또 다른 차량에 대한 공격 벡터가 되어 여러 CAN 패킷을 활용한 공격 등이 발표되는 계기가 되었다[1]. 따라서 이러한 CAN 패킷에 의한 공격을 방어하는 것은 공격에 의해 위협받는 운전자의 생명을 보호하는 데 있어 중요한 역할을 한다. 본 문서는 정상적인 상태의 차량의 CAN 패킷을 분석한 결과를 바탕으로 특정한 CAN 패킷들이 공격 패킷인지에 대한 여부를 판단한다.

II. 방법론

팀 "카놀라유"에서 제시한 CAN 패킷 공격 탐지 기술은 Survival Rate 기법, Hamming Distance 기법 및 Loose Ground Truth 기법으로 생성된 공격 및 정상 패킷 판정을 비교하여 우세한 결과를 최종 판정으로 정하는 기법이다. 각 방법에 대한 설명은 다음과 같다.

1. Survival Rate 기법

Survival Rate 기법은 의료분야에서 널리 쓰이는, 예를 들어 어떤 약을 투약했을 때 특정 시점에서 환자의 생존 여부를 확률적으로 계산하는 등의, Survival Rate를 활용하여 CAN 패킷을 이용한 공격을 탐지하는 기법을 발표한 2018년 논문에 대한 구현[2]을 그대로 이용한다. 어떠한 CAN ID에 대해 Survival Rate를 계산하는 방법은 아래 그림과 같다.

$$SR(C_i^{CAN\_ID}) = \frac{F(C_i^{CAN\_ID})}{S(C_i^{CAN\_ID})} \quad (1)$$

- $C_i = \{C_1, C_2, \ldots, C_i \parallel 1 \le i \le n\}$: 특정 시간대의 CAN msg 들을 특정한 수로 나눈 Chunk들
- i: Chunk의 index
- $S(C_i^{CAN\_ID})$: 하나의 Chunk unit에 대한 모든 CAN msg
- $F(C_i^{CAN\_ID})$: 하나의 Chunk unit에서 특정 CAN msg가 출연하는 빈도
- $SR(C_i^{CAN\_ID})$: 하나의 Chunk unit에서 특정 CAN msg의 Survival rate

Survival Rate 기법은 먼저 공격이 없는 CAN 패킷을 활용하여 CAN ID별 Survival Rate의 MIN 값 및 MAX 값을 구한다. 이렇게 구한 Survival Rate의 값으로 미지의 패킷들

의 CAN ID별 Survival Rate를 계산한 다음, 이 Survival Rate가 MIN 및 MAX 범위를 벗어나면 공격 패킷으로, 그렇지 않으면 정상 패킷으로 판정한다. 논문에서 설명하고 있는 Anomaly detection 알고리즘은 다음과 같다.

Algorithm 2: Anomaly detection module.

```
   Input: G (ground-truth value)
   Output: alert message
1  calling the survival rate operation module
2  M ← returning survivel rate of CAN IDs
3  for i ∈ {0, ..., M} do
4      for j ∈ {0, ..., G} do
5          count ← counting the M[i][0] among the G[j][0]  /* The CAN ID
           in the two-dimensional array */
6          if count ≥ 1 then
7              if G[j][0] ≠ M[i][0] then
8                  continue
9              else
10                 if G[j][1] ≤ M[i][1] then       /* Comparison of the
                   ground-truth value with the survival rate
                   of CAN ID */
11                     if G[j][2] ≥ M[i][1] then
12                         flag[M[i][0]] ← 0
13                     else
14                         output the alert message
15                         flag[M[i][0]] ← 1
16                     end
17                 else if G[j][2] ≥ M[i][1] then
18                     if G[j][1] ≤ M[i][1] then
19                         flag[M[i][0]] ← 0
20                     else
21                         output the alert message
22                         flag[M[i][0]] ← 2
23                     end
24                 end
25             end
26         end
27     end
28 end
```

Survival Rate 기법의 단점은 두 가지가 있다. 첫 번째로, 공격 패킷 판별 여부를 Chunk 단위로 수행하기 때문에, 어떠한 CAN ID의 Survival Rate가 해당 Chunk 안에서 공격이나 정상으로 판정되면 각각의 CAN ID의 공격 패킷 여부와는 상관없이 그 Chunk 안에서의 해당 CAN ID 패킷은 모두 한 가지의 결과로 고정된다. 또한 Chunk의 크기가 고정되어있기 때문에 출현 빈도가 높은, 즉 Cycle이 긴, CAN 패킷에 대한 판정 자체가 어렵다.

2. Hamming Distance 기법

Hamming Distance는 같은 길이를 가진 두 개의 문자열의 서로 다른 문자 개수를 의미하며, Hamming Distance를 활용한 CAN 침입 탐지 기법[3]을 차용하여 침입탐지에 활용한다. Hamming Distance에 대한 공식은 다음과 같다.

$$\mathcal{H}_d(p_t, p_{t+1}) = \sum_{i=1}^{64} p_t^i \otimes p_{t+1}^i$$

Pt는 시간 t에서의 CAN 패킷의 페이로드이며, Pti는 해당 페이로드의 i번째 비트를 뜻한다. Hd는 CAN ID d에서의 Hamming distance로, d에 해당하는 CAN ID의 시간 t와 시간 t+1 사이의 모든 비트의 XOR 연산 결과에서의 1의 총 개수를 뜻한다. Hamming Distance 기법을 이용한 공격 패킷의 판정은 ID별 Hamming distance에 대한 Max, Min값을 측정하여 기

록, 탐지 시 Hamming distance의 값이 Max, Min 범위 안이면 정상, 범위 밖 공격으로 탐지한다. Hamming Distance 기법의 단점은, Hamming Distance의 MIN값과 MAX값의 차이가 지나치게 큰 경우, 논문에서는 6 이상인 경우로 서술, 탐지 정확도가 급락한다는 점이다. 이를 해결하기 위해서 Hamming Distance가 정상적으로 판단되도록, 데이터 필드 중 값이 변하지 않는 바이트의 인덱스를 확인하는 방식을 적용하였다.

3. Loose Ground Truth 기법

Ground Truth 기법은 정상적인 CAN 패킷들에서 CAN ID 별 Data field의 각각 Byte index에 올 수 있는 값을 산정하고, 이를 Ground Truth로 지정하는 기법이다. 임의의 CAN 패킷들을 하나씩 비교하여 그 패킷이 Ground Truth 범위 내에 포함되는 패킷이라면 정상, 그렇지 않을 경우 공격으로 탐지한다.

Loose Ground Truth 기법에서는 Ground Truth 기법을 확장하여 데이터 필드의 인덱스를 고려하지 않고 등장하는 바이트만을 고려한다. 테스트 데이터에서 차종별로 'R' 플래그에 대해 CAN ID에 해당하는 데이터 필드 안의 모든 바이트를 담는다. 그 후 제출용 데이터셋에서 차종별로 없었던 CAN ID인 경우 공격 패킷으로 판정하고, 존재한 CAN ID인 경우 데이터 필드가 등장하지 않았던 바이트라면 공격 패킷으로 판정하고, 모든 바이트가 등장했다면 정상 패킷으로 판정한다.

Loose Ground Truth 기법에서는 과거의 CAN 패킷들에 대해서만 고려를 하므로, 새롭게 등장하는 CAN 패킷들에 대해서는 오탐할 가능성이 높다는 한계점이 있다.

4. RNN-LSTM 모델 학습 기법

본 기법은 제출 시점에 반영되어있지 않은 방법이며, RNN의 일종인 LSTM 신경망 모델을 사용하는 기법으로, Word Vector 학습을 통한 Binary Sentiment Classification 관련 논문에서의 방법론을 적용하였다. [4]TRAIN 데이터에서 CAN ID와 Data field, Byte Index를 입력 데이터, 제출용 데이터의 CAN ID 와 Data field를 출력 데이터로 설정한다. 그리고 순환 신경망에서의 학습을 통해 제출용 데이터의 CAN ID와 Data field를 고려하여 Byte Index를 예측한다.

LSTM 신경망 학습 기법의 경우, test set에 대해서도 답이 존재해야 학습이 가능하다. 따라서 제출용 데이터가 모두 정상 패킷이라는 가정 하에 진행한 학습을 바탕으로 최종 결과(공격 여부)를 도출하므로 정확도가 다소 떨어질 수 있다는 단점이 있다.

III. 탐지결과 및 평가

1. Survival Rate 기법

Survival Rate 기법을 이용하여 테스트용 데이터셋과 비교한 정확도를 측정하였다. Spark와 Sonata, Soul의 정확도는 다음과 같다.

- Spark

  Detection rate(flooding): 94.424706%

  Detection rate(fuzzy): 98.097561%

  Detection rate(malfunction): 97.866667%

- Sonata
  Detection rate(flooding): 81.114583%
  Detection rate(fuzzy): 90.454545%
  Detection rate(malfunction): 92.312644%

- Soul
  Detection rate(flooding): 85.461667%
  Detection rate(fuzzy): 80.695833%
  Detection rate(malfunction): 98.240708%

2. Hamming Distance 기법

Survival Rate 기법에서와 같이 테스트용 데이터셋과 비교한 정확도를 측정하였다. Spark와 Sonata, Soul의 정확도는 다음과 같다.

- Spark
  Detection rate(flooding): 100%
  Detection rate(fuzzy): 100%
  Detection rate(malfunction): 100%

- Sonata
  Detection rate(flooding): 100%
  Detection rate(fuzzy): 100%
  Detection rate(malfunction): 64%

- Soul
  Detection rate(flooding): 100%
  Detection rate(fuzzy): 100%
  Detection rate(malfunction): 100%

3. Loose Ground Truth 기법

Survival Rate 기법에서와 같이 테스트용 데이터셋과 비교한 정확도를 측정하였다. Spark와 Sonata, Soul의 정확도는 다음과 같다.

- Spark
  Detection rate(flooding) : 100.0%
  Detection rate(fuzzy) : 99.983566%
  Detection rate(malfunction) : 100%

- Sonata
  Detection rate(flooding) : 100.0%
  Detection rate(fuzzy) : 99.591521%

Detection rate(malfunction) : 66.666667%

- Soul
  Detection rate(flooding) : 100.0%
  Detection rate(fuzzy) : 99.967602%
  Detection rate(malfunction) : 100.0%

4. RNN-LSTM 모델 학습 기법

Survival Rate 기법에서와 같이 테스트용 데이터셋과 비교한 정확도를 측정하였다. Spark와 Sonata, Soul의 정확도는 다음과 같다. (epoch=3, batch size=64 기준)

- Spark
  Detection rate(flooding) : 100.0%
  Detection rate(fuzzy) : 85.16%
  Detection rate(malfunction) : 84.34%

- Sonata
  Detection rate(flooding) : 100.0%
  Detection rate(fuzzy) : 83.33%
  Detection rate(malfunction) : 82.58%

- Soul
  Detection rate(flooding) : 80.21%
  Detection rate(fuzzy) : 90.63%
  Detection rate(malfunction) : 93.91%

IV. 프로그램 설명

모든 기법들은 Python으로 구현되었다.

1. Survival Rate 기법

forza_anomaly 디렉토리에서 테스트 데이터들을 ./dataset/training 에, 제출용 데이터들을 ./dataset/qualifying 에 위치시킨다.

1. forza_anomaly.py : 주어진 Training data set을 기반으로 Survival Rate를 산출한 다음, 이를 바탕으로 제출용 데이터의 공격 패킷 여부를 판정한다.

forza_anomaly.py를 실행하면 판정이 진행된다. (python forza_anomaly.py)

2. Hamming Distance 기법

rule_base 디렉토리에서 테스트 데이터들을 ./testset에, 제출용 데이터들을 ./final에 위치시킨다.

1. hamming_dict_maker.py : 주어진 training data set을 기반으로 id, dlc, payload data, hamming distance에 관한 rule을 생성 함수 모듈.

2. hamming_detector.py : 생성된 rule을 기반으로 탐지 대상 data에 대한 탐지를 실행하여 csv로 출력.

hamming_detector.py를 실행하면 판정이 진행된다. (python hamming_detector.py)

3. Loose Ground Truth 기법

feature_base 디렉토리에서 테스트 데이터들을 ./testset에, 제출용 데이터들을 ./final에 위치시킨다.

1. feature.py : testset에 있는 파일들에서 차종별로 정상 패킷만을 고려하여 CAN ID와 이에 해당하는 데이터 필드의 값을 dictionary로 반환한다.

2. correction_main.py : 패킷별 CAN ID가 feature.py에서 추출한 dictionary에 있는지 확인하고, 없으면 공격 패킷으로 분류한다. 또한 CAN ID가 있어도, 데이터 필드의 모든 바이트가 추출한 dictionary에 존재하지 않는다면 공격 패킷으로 분류한다.

correction_main.py을 실행하면 판정이 진행된다. (python correction_main.py)

4. RNN-LSTM 기법

본 기법은 제출 시점에 반영되어있지 않은 방법으로, RNN의 모델 중 LSTM을 사용한 기계학습으로 CAN 패킷의 공격 패킷 여부를 판정한다. 본 기법의 적용은 keras 라이브러리를 사용해 LSTM 모델을 구현하는 것으로 구현하였다. CAN 패킷들에 대한 판정 방법은 여러 기법의 적용 결과 기계학습 계열의 방법론보다 패킷 해석 계열의 방법론이 더 효과적인 것으로 결론지어졌기 때문에 제출 데이터의 판정에 사용되지 않은 기법이다.

keras_lstm_detection 디렉토리에서 테스트 데이터들을 ./testset에, 제출용 데이터들을 ./final에 위치시킨다.

1. test.py : TRAIN 데이터들을 파일 단위로 50%는 train set, 나머지 50%는 test set으로 설정한다. CAN ID, 데이터, 공격 여부만을 학습 데이터로 사용하는데, CAN ID와 데이터는 병합하여 payload 리스트로, 공격 여부는 단독으로 받아 flag 리스트로 만든다. lstm_learning 함수에서는 train set의 payload(CAN ID와 데이터) 를 X_train, flag를 Y_train으로 설정하고, 같은 방법으로 test set 역시 각각 X_train 과 X_test로 설정한다. 현재 상태에서 CAN ID와 데이터는 하나의 문자열의 형태로 저장되어 있으므로 LSTM 모델에서 학습할 수 있도록 토큰화한다. 또한 X_Train과 X_test에 있는 CAN ID와 데이터를 병합한 payload의 길이가 하나의 일정한 값을 갖지 않으므로, sequence padding을 통해 각 payload의 길이가 일정하도록 전처리한다. Embedding을 통해 토큰화 된 데이터를 다시 밀집 벡터로 변환하고 lstm 모델을 통해 학습시키고, 최종적으로 탐지율을 산출한다

2. rnn.py : 코드의 기본적인 컨셉은 test.py와 동일하다. 단, rnn.py 에서는 test.py와는

달리 TRAIN 데이터는 모두 train set으로, 제출용 데이터를 test set으로 설정한다. 이 경우 train set과 test set의 데이터 크기가 다를 수 있으므로, 크기가 더 큰 쪽에 맞춰 데이터 개수를 조정한다. (크기가 작은 쪽의 마지막 Index에 있는 데이터-CAN ID, 데이터, 공격 여부- 를 두 파일의 크기가 같아질 때 까지 복제하여 저장한다. 첫 번째 실행의 경우 제출용 데이터에서의 각 패킷의 공격 여부를 알 수 없으므로, 임의로 제출용 데이터는 모두 정상 패킷인 것으로 간주하여 LSTM 신경망에서 학습시킨다. 학습이 완료되면 학습 내용을 바탕으로 predict_class를 통해 최종 결과에 대한 예측을 제시하며, 이 예측값이 해당 Payload(CAN ID와 데이터)에 대한 공격 여부이다.

V. 결론

세 가지의 기법을 적용하여 우세한 결과를 최종 결과로 산출하였다. 예를 들어 Survival Rate와 Hamming Distance에서 T로 판정하고, Loose Ground Truth 기법에서 R로 판정한 CAN 패킷은 결과적으로 T로 판정한다.

[참고문헌]

[1] Miller, Charlie, and Chris Valasek. "Remote exploitation of an unaltered passenger vehicle." Black Hat USA 2015 (2015): 91.

[2] Mee Lan Han, Byung Il Kwak, and Huy Kang Kim. "Anomaly intrusion detection method for vehicular networks based on survival analysis." Vehicular Communications 14 (2018): 52-63.

[3] Stabili, Dario, Mirco Marchetti, and Michele Colajanni. "Detecting attacks to internal vehicle networks through Hamming distance." 2017 AEIT International Annual Conference. IEEE, 2017.

[4]Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. "Learning Word Vectors for Sentiment Analysis" Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1, 142-150, 2011