APIコールの時系列情報に注目したLSTMスタッキングによるマルウェアファミリの早期分類

丸若 弘介[1,a)]　青木 茂樹[1,b)]　宮本 貴朗[1]

概要：近年のサイバー攻撃の巧妙化に伴って, マルウェアの侵入を防ぐことが難しくなっている. マルウェア侵入後の被害を最小限に抑えるためには, 侵入したマルウェアのマルウェアファミリを迅速に推定することが重要である. そこで本稿では, マルウェアの API コールの時系列情報に着目してマルウェアファミリを早期に推定する手法を提案する. まず, 観測された API コールログ中の関数名を時系列順に抽出し, 先頭から, 30%, 50%, 70%, 90%, 100%の長さで分割して, 長さごとに異なる LSTM で学習する. 次に, 学習したそれぞれの LSTM の出力結果を統合して学習し, メタモデルを構築する. その後, 新たな API コールの時系列情報が観測された時, マルウェア侵入後の初期段階のみを学習した LSTM を含む, 全ての LSTM で時系列情報を出力させてメタモデルに入力し, マルウェアファミリを早期に推定する. 実験では, Soliton Dataset2020, 2021 を用いてマルウェアファミリを推定し, 有効性を確認した.

キーワード：マルウェアファミリ早期分類, LSTM, スタッキング

Early Classification of Malware Families Using LSTM-Based Stacking Focused on API Call Time-Series Information

Kosuke Maruwaka[1,a)]　Shigeki Aoki[1,b)]　Takao Miyamoto[1]

Abstract: In recent years, the increasing sophistication of cyber attacks has made it increasingly difficult to prevent malware intrusion. To minimize the damage after infiltration, it is crucial to promptly estimate the characteristics of the malware family. This paper proposes a method for early malware family classification by leveraging the time-series information of API call logs. Specifically, API function names are extracted in chronological order and segmented into portions representing 30%, 50%, 70%, 90%, and 100% of the log length. Each segment is used to train a separate LSTM model. The outputs of these LSTM models are then integrated and used as input features for a meta-model constructed using gradient boosting. During inference, the observed API call sequence—regardless of its length—is processed by all LSTM models, and their outputs are fed into the meta-model to estimate the malware family at an early stage. Experiments conducted using the Soliton Dataset 2020 and 2021 demonstrate that the proposed method achieves high classification accuracy across varying lengths of input data, confirming its effectiveness and robustness in realistic scenarios.

Keywords: Early classification of malware families, LSTM, Stacking Ensemble

1. はじめに

近年のマルウェアの巧妙化に伴って, 侵入時の検知が難しくなってきている [1]. そのため, マルウェアの侵入を前提として, 検知したマルウェアの特性を迅速に特定することにより被害の拡大を防ぐことが求められている. 実環境

[1] 大阪公立大学大学院情報学研究科
Graduate School of Informatics, Osaka Metropolitan University
[a)] sd24713y@st.omu.ac.jp
[b)] aoki@omu.ac.jp

に導入されているセキュリティ対策は, マルウェア感染を防止する EPP(Endpoint Protection Platform) とマルウェア感染後の対応を支援する EDR(Endpoint Detection and Response) の二層構造で構成されている. EDR には, 検知, 隔離, 調査, 復旧までのプロセスがある. そして, 検知と隔離までのプロセスを自動化し, 被害が出る前にマルウェアを隔離する製品が実用化されている.

一般にマルウェアはいくつかのマルウェアファミリに分類されており, マルウェアファミリ毎に類似するパターンで挙動することが多い. マルウェアの挙動に注目してマルウェアファミリを早期に特定することができれば, マルウェアファミリに応じた対策を講じることができ, 侵入後の被害を低減できると考えられる. 例えば, 侵入したマルウェアのマルウェアファミリがランサムウェアである場合は, ネットワークの切断とバックアップの取得, 外付けディスクの切り離しなど, ランサムウェアに特化した対策を早期に実施することで, 被害を最小限に抑えることができると考えられる.

マルウェアファミリを特定する際に用いる情報の一つとして, API コールログ情報がある. API コールログ情報とは, ソフトウェアが別のソフトウェアの機能やデータを呼び出して利用した履歴である. 同一のマルウェアであれば機能やその実装方法は類似していると考えられるため, マルウェアファミリ毎に API コールログ情報は類似していると考えられる. API コールログ情報からマルウェアファミリを分類する代表的な研究として, 文献 [2], [3], [4], [5] の手法が挙げられる. 文献 [2] では, API コールログ中の関数名の出現回数を特徴ベクトルとし, ランダムフォレストの回帰モデルを用いて将来の挙動を予測する手法を提案している. この手法では関数の出現回数に着目することで, マルウェアの持つ機能の予測精度は向上しているものの, 時系列情報には注目していない. API コールの時系列情報に着目することで, 更に早期にマルウェアファミリを推定できる可能性が高くなると考えられる. API コールログ情報の時系列に注目した手法 [3] では, API コールログ中の関数名の時系列情報を LSTM(Long Short Term Memory) と GRU(Gated Recurrent Unit) で学習してマルウェアファミリを分類している. また文献 [4] では, 文献 [3] で用いられた時系列情報を注意機構付き LSTM で学習してマルウェアファミリを分類している. これらの手法では, API コールログの時系列情報に注目して高精度にマルウェアファミリを分類できているものの, 早期分類に主眼を置いていないため, 早期分類性能については言及されていない. API コールログの時系列情報に注目し, 早期分類にも主眼を置いた文献 [5] では, API コールログの時系列情報を複数の LSTM で学習し, 各 LSTM の予測結果をルールベースで統合してマルウェアファミリを分類している. この手法では, API コールログを複数の部分系列 (先頭から 30%, 50%など) に分割して, それぞれを個別の LSTM で学習することで, 感染初期の API コールログ情報の特徴を十分に捉えられることを確認している. しかし, 個別の LSTM の出力を単純なルールで統合しているために, それぞれの LSTM の利点を活かした最適な統合方法となっていないことが課題となっていた.

そこで本研究では, 文献 [5] の各 LSTM の統合方法を最適化することにより, 新たな API コールの時系列情報が得られた時にマルウェアファミリを早期に分類する手法を提案する. 具体的には, マルウェアの挙動段階ごとの API コールの時系列情報をそれぞれ LSTM で学習し, 各 LSTM をベースモデルとしてスタッキングする. スタッキングは, 複数のモデルを組み合わせて予測精度を向上させるアンサンブル学習手法である. 本手法では, 感染初期にマルウェアを推定することができるため, EDR に応用することでマルウェアファミリに対応した最適な対策を早期に開始し, 被害を最小限に抑えることができると考えられる. 以下, 第 2 節では関連研究について説明する. 第 3 節で提案手法について, 第 4 節では実験と結果に対する考察を述べ, 第 5 節で結びとする.

2. 関連研究

本研究に関連する従来研究として, API コールログ情報から LSTM と GRU を用いてマルウェアファミリを分類する手法 [3] と, API コールログ情報から 2 つの深層学習手法を組み合わせてマルウェアを検出する手法 [6], マルウェアの API コールの時系列情報の部分系列を各 LSTM で学習してルールベースで統合してマルウェアファミリを分類する手法 [5] について述べる. また, 近年一般的となった深層学習ベースのスタッキングをマルウェア検出に応用した研究として, 文献 [7] について述べる.

文献 [3] では, API コールログ情報の時系列から LSTM と GRU を利用してマルウェアファミリを分類する手法を提案している. この手法ではまず, API コールログ情報から関数名を抽出し, 抽出した関数名が連続して重複したり, 同じパターンで繰り返されるような冗長性があるものを削除する. 次に, 冗長性を削除した API コール列をワンホットベクトルに変換し, LSTM と GRU でそれぞれ学習する. その後, 学習した LSTM と GRU を用いて新たな API コールログ情報に含まれるマルウェアファミリを分類している. この手法では API コールログ情報の時系列に注目しているものの, マルウェアファミリの早期分類に主眼を置いた研究ではないために, 早期分類時の性能については言及されていない. 文献 [6] では, API コールログ情報から CNN と BiGRU を組み合わせてマルウェアを検出する手法を提案している. この手法ではまず, API コールログ情報をエンコーディングして数値変換し, ベクトル化する. その後, CNN で局所的なパターンを見つけ出し, BiGRU でパター

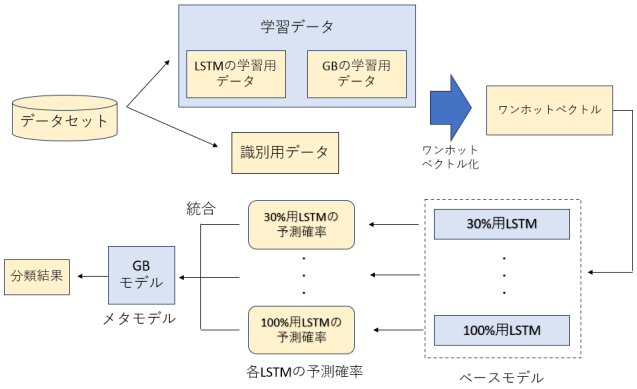
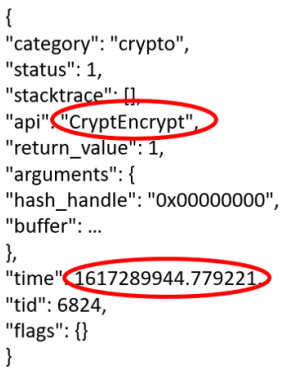


図 1: 提案手法の概要

Fig. 1 Overview of the proposed method.



```
{
"category": "crypto",
"status": 1,
"stacktrace": [],
"api": "CryptEncrypt",
"return_value": 1,
"arguments": {
"hash_handle": "0x00000000",
"buffer": ...
},
"time": 1617289944.779221,
"tid": 6824,
"flags": {}
}
```

図 2: API コールログの例

Fig. 2 Example of API call.

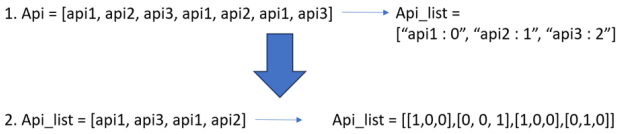


図 3: ワンホットベクトル化の手順

Fig. 3 One-hot encoding procedure for API calls.

ン間の時間的な関係性を学習する．この手法では 2 つの深層学習手法を組み合わせることで単一の深層学習手法を用いる場合よりも高精度に検出できている．しかし，テストデータの系列長が 100 の場合と系列長が 20 の場合の実験結果を比較すると，検知精度が 10%以上低下している．この結果から，十分な長さの時系列情報が必要であり，初期挙動のみでマルウェアを検出することが難しいことを確認している．文献 [5] では，API コールログの時系列情報を複数の部分系列に分割して，それぞれを個別の LSTM で学習してマルウェアファミリを早期検知し，分類する手法を提案している．この手法ではまず，API コール情報から関数名を取り出し，ワンホットベクトルに変換する．次に，API コール列を先頭から 30%, 50%, 70%, 90%, 100%の長さで分割して，長さごとに異なる LSTM でそれぞれ学習する．その後，新たな API コール列が得られた時，5 種類全ての LSTM に入力し，それぞれの出力結果を統合する．統合方法として，各 LSTM で出力した確率分布の総和が最大のマルウェアファミリを推定結果とする方法と，各 LSTM で出力した確率分布の最高値のマルウェアファミリを推定結果とする方法を用いている．この手法では単純なルールベースでの統合のため，最適な統合方法とはなっていないことが課題となっていた．

文献 [7] では，Android マルウェア検出におけるロバスト性を向上させるために，3 つの深層学習手法を統合したスタッキングによるマルウェア検出手法を提案している．この手法では 3 つの深層学習の出力結果を特徴量とした別の学習器 (メタモデル) を用いて高精度な検出を実現している．

3. 提案手法

本手法の概要を図 1 に示す．まず，データの前処理の段階では，準備したデータセットを学習に使用する部分と，モデル全体の評価に使用する部分に分割する．学習データは，ベースモデルである LSTM とメタモデルである Gradient Boosting(GB) の学習用に分割する．そして，API コール時系列情報をワンホットベクトルに変換する．次にワンホットベクトル化した学習用データを先頭から 30%, 50%, 70%, 90%, 100%の長さで分割し，それぞれの長さに対応した LSTM で学習し，スタッキングのベースモデルとする．各ベースモデルの学習後，GB の学習用データを 5 つのベースモデルの LSTM に入力し，それぞれの LSTM で予測確率を出力して統合する．統合したものをメタ特徴量としてメタモデルの GB で学習する．その後，新たな API コール時系列が観測された時，全てのベースモデルの LSTM からの出力結果を統合してメタ特徴量を生成し，メタモデルの GB でマルウェアファミリを識別する．

3.1 特徴量抽出

まず，API コールログ中の関数名を時系列順に抽出する．ログファイルの一部を抜粋したものを図 2 に示す．ログファイルには API 単位の詳細な挙動が記録されている．赤丸で示している部分が上から API の関数名と Unixtime である．Unixtime は 1970 年 1 月 1 日午前 0 時 0 分 0 秒からの経過秒数で，関数名を時系列順に並べる際に利用する．次に，API コール列をワンホットベクトルの系列に変換する．変換手順を図 3 に示す．まず，取得した全ての API コールから辞書を作成する．その後，辞書化したリストを基にワンホットベクトルに変換する．ここでワンホットベクトルとは，ある次元数のベクトルの内，一つの要素だけが 1 で残りの要素が全て 0 のベクトルであり，ここでは辞書内の単語の順序を表す要素を 1 としたワンホットベクトルに変換している．その後，マルウェアファミリが異なる場合やマルウェアの進化により変化するデータ長を揃えるため，

図 4: パディングによる系列長正規化

Fig. 4 Sequence length normalization using padding.

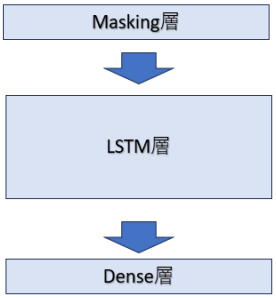


図 5: 分類モデル構造

Fig. 5 Structure of the classification model.

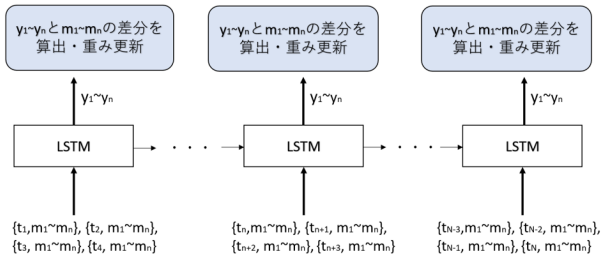


図 6: LSTM による学習処理の例

Fig. 6 Example of the LSTM-based learning process.

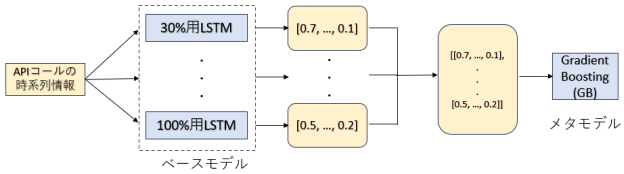


図 7: メタモデル入力までの流れ

Fig. 7 Workflow leading to meta-model input.

パディングする．パディングの例を図 4 に示す．最も長い API コール列の系列長に合わせるように短い API コール列を 0 で埋めることにより，系列長をそろえる．

3.2 API コール列の LSTM(ベースモデル) による学習

LSTM を用いた分類モデルの構造を図 5 に示す．モデル構造は，3 層で構成されている．1 層目の Masking 層では，学習の際にパディングにより加えたゼロベクトルの影響が出ないようマスクをかけ，学習器に無視させる．2 層目の LSTM 層では，入力データの時間的なパターンや依存関係を学習し，最終的なクラスを予測する．3 層目の Dense 層では，softmax 関数を使用して，入力データがどのクラスに属する可能性が高いかを示す確率分布を出力する．

API コール列を先頭から 30%，50%，70%，90%，100%の長さで分割して，長さごとに異なる LSTM でそれぞれ学習する．学習では，API コール列のワンホットベクトルの系列を説明変数，ワンホットベクトルの系列に対応するマルウェアファミリのワンホットベクトルを目的変数として設定する．学習の流れを図 6 に示す．まず，説明変数 t_n，t_{n+1}，t_{n+2}，t_{n+3} と対応する正解ラベルの目的変数 m_1 ～ m_n を入力し確率分布 y_1 ～ y_n を出力する．その後，確率分布 y_1 ～ y_n と目的変数 m_1 ～ m_n の差分を算出し，算出した差分を基に LSTM の重みを更新する．以上の処理を繰り返し，事前に定めたエポック数に達したら学習を終了する．以上の処理で学習した長さごとの LSTM をベースモデルとする．

3.3 GB(メタモデル) による学習とマルウェアファミリの識別

ベースモデルである各 LSTM の出力からメタ特徴量を生成し，メタモデルに入力するまでの流れを図 7 に示す．API コールの時系列情報をベースモデルの全ての LSTM に入力し，それぞれ 1 次元の予測確率分布を出力する．そして，5 つの確率分布を統合した 5 次元のベクトルをメタ特徴量とし，メタモデルである GB に入力して単一の予測ラベルを出力する．GB の学習は，メタ特徴量を説明変数，マルウェアファミリのワンホットベクトルを目的変数として設定する．

新たな API コール列が得られた時，ベースモデルの全ての LSTM に入力して得られたそれぞれの予測確率を統合してメタ特徴量である 5 次元ベクトルを抽出する．その後，メタモデルに入力して識別結果を出力する．

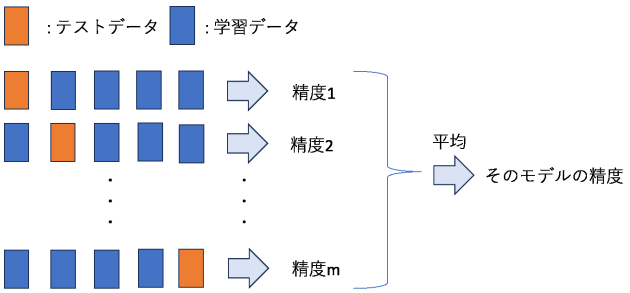


図 8: Leave-One-Out 交差検証の概要

Fig. 8 Overview of the Leave-One-Out Cross-Validation procedure.

4. 実験

4.1 実験条件

データセットとして，MWS Datasets[9], [10] の一部として提供されている Soliton Dataset 2020, 2021 を使用した．Soliton Dataset とはエンドポイントセキュリティ製品である InfoTrace Mark II for Cyber[11] （以下，Mark II と呼

表 1: マルウェアファミリ名とラベル
Table 1 Malware family names and corresponding labels.

マルウェアファミリ名	ラベル
NetWire	0
RoyalRoad	1
IcedID	2
LazyScripterA	3
Upatre	4
Predator the Thief	5
Trickbot	6
Bazar	7
COPPERHEDGE	8
Valyria	9
STOP Ransomware	10
Minerdropper	11
Evasive Azorult	12
USBferry	13
Emotet	14

ぶ）が導入された環境でマルウェアを実行して得られた動的解析ログを中心としたデータセットである. Cuckoo 上で Mark II を導入したゲスト環境を使用して, 1 検体につき Mark II ログと Cuckoo ログの 2 つが生成されている. その 2 つの内, API 単位の詳細な挙動が記録される Cuckoo ログを利用して実験を行った. 実験には, 15 種類のマルウェアファミリを各 20 ファイルずつ, 計 300 個の検体分のデータを使用した. 使用した検体のマルウェアファミリ名とラベルを表 1 に示す.

分類評価手法として, クロスバリデーションの一種である LOOCV(Leave-One-Out Cross Validation)[8] を用いた. 評価方法の概要を図 8 に示す. LOOCV は一つだけをテストデータとし, 残り全てのデータを学習データとして評価する手法である. そして, テストデータを変更して算出した全評価の平均を最終的な性能とする.

データセットのデータは, 検体ごとに API コールの時系列の系列長が異なっており, 先頭から, 30%, 50%, 70%, 90%, 100%の長さのものを使用して, 本手法で早期にマルウェアファミリを分類できることを確認する実験を行った. また, 本手法の有効性を示すため, 文献 [5] の実験結果と比較した.

4.2 実験結果と考察

ベースモデルにおける単一の LSTM による分類結果と本手法の結果を表 2 に示す. 単一の LSTM を用いた場合, 学習時と同じ長さのテストデータに対しては, すべてのモデルで Accuracy が 0.98 以上と高精度であった. しかし, 学習時とは異なる長さのテストデータに対しては, 分類精度が大きく低下した. 一方本手法では, テストデータの長さにかかわらず, すべてのケースで Accuracy が 0.93 以上を維持しており, 高い分類性能を示した. テストデータの全長が既知である場合には, 単一の LSTM モデルでも高精度な分類が可能であるが, 実環境ではテストデータの全長が未知であることが多く, どの長さを学習した LSTM を用いるべきか判断ができない. 本手法では, テストデータの長さに依存せず高精度な分類が可能であるため, 単一の LSTM

表 2: 提案手法とベースモデル毎の LSTM による マルウェアの早期分類結果
Table 2 Early classification results using LSTM for each proposed and base model.

テストデータ長	学習した長さ	ベースモデルの Accuracy	提案手法の Accuracy
30%	30%	0.990	0.950
	50%	0.997	
	70%	0.783	
	90%	0.690	
	100%	0.700	
50%	30%	0.990	0.963
	50%	0.997	
	70%	0.783	
	90%	0.690	
	100%	0.700	
70%	30%	0.827	0.933
	50%	0.727	
	70%	0.990	
	90%	0.757	
	100%	0.777	
90%	30%	0.747	0.963
	50%	0.660	
	70%	0.753	
	90%	0.987	
	100%	0.790	
100%	30%	0.706	0.940
	50%	0.620	
	70%	0.730	
	90%	0.730	
	100%	0.983	

表 3: 提案手法と先行研究の早期分類結果
Table 3 Early classification results of the proposed method and previous research.

長さ	先行研究 (総和の最大値)	先行研究 (確率分布の最高値)	提案手法
30%	0.72	0.89	0.950
50%	0.89	0.92	0.963
70%	0.94	0.93	0.933
90%	0.94	0.92	0.963
100%	0.87	0.92	0.940

モデルを用いた手法よりも実環境において有効性が高いことを確認できた．

本手法と先行研究 [5] の比較結果を表 3 に示す．提案手法は系列長が 70%の場合を除き，すべてのテストデータ長において先行研究より高い Accuracy を示しており，高精度な分類が可能であることを確認できた．この結果から，スタッキング手法を用いることで，各 LSTM の予測結果をより効果的に統合できたために，先行研究の統合手法（総和の最大値）よりも優れた分類性能を実現できたと考えられる．一方，70%のデータ長においては，提案手法の Accuracy が先行研究よりわずかに低下している．原因としては，70%の時点における API コールログに異なるマルウェアファミリ間で類似性のある挙動が含まれていたために，誤分類が起きた可能性が考えられる．今後，誤分類の要因を詳細に調査したいと考えている．

テストデータの長さが 70%，90%，100%の場合における各マルウェアファミリの分類結果の混同行列を，表 4，表 5，表 6 にそれぞれ示す．表に示す結果から，特にマルウェアファミリ「1」と「11」において分類結果に差異が見られた．ファミリ「1」では，70%のテストデータに対して 14 件が正しく分類され，4 件がファミリ「11」，2 件がファミリ「12」に誤分類された．90%では 16 件が正しく分類され，1 件がファミリ「11」，3 件がファミリ「12」に誤分類された．100%では 13 件が正しく分類され，4 件がファミリ「11」，3 件がファミリ「12」に誤分類された．一方，ファミリ「11」では，70%のテストデータに対して 12 件が正しく分類され，1 件がファミリ「1」，7 件がファミリ「12」に誤分類された．90%では 19 件が正しく分類され，1 件がファミリ「12」に誤分類された．100%では 15 件が正しく分類され，1 件がファミリ「1」，4 件がファミリ「12」に誤分類された．これらの結果から，70%および 100%の時点では，ファミリ「1」，「11」，「12」の間で挙動に類似性がある可能性が考えられる．今後の改善策としては，API コールの関数名だけではなく，引数や戻り値などの詳細な情報を特徴量として活用することで，類似した挙動の区別が可能になると考えられる．また，注意機構（Attention Mechanism）を導入し，重要度の低い API コールを無視させることによって，分類精度の向上を図ることも有効であると考えられる．

5. まとめ

本稿では，API コールの時系列情報を複数の長さに分割し，分割した各部分系列を個別の LSTM モデル（ベースモデル）で学習させた後，それぞれの予測確率を統合して GB（メタモデル）に入力して識別するマルウェアファミリの識別手法を提案した．実験の結果，部分系列の長さに依存せず，高精度にマルウェアを識別できることを確認した．実験結果より，実環境においても早期に安定した分類性能でマルウェアを識別できると考えられ，EDR に応用することでマルウェアに対応した最適な対策を早期に開始し，被害を最小限に抑えることができると考えられる．

今後の課題としては，70%および 100%のデータ長において誤分類が生じた原因と考えられる，類似性のある挙動への対応が挙げられる．具体的には，API コールの関数名だ

表 4: 70%の長さを識別した際の混同行列
Table 4 Confusion matrix for 70% sequence length.

正解ラベル	予測ラベル														
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	14	0	0	0	0	0	0	0	0	0	4	2	0	0
2	0	0	19	0	0	0	0	1	0	0	0	0	0	0	0
3	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	20	0	0	0	0	0	0	0	0
7	0	0	2	0	0	0	0	18	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0
11	0	1	0	0	0	0	0	0	0	0	0	12	7	0	0
12	0	2	0	0	0	0	0	0	0	0	0	1	17	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20

表 5: 90%の長さを識別した際の混同行列
Table 5 Confusion matrix for 90% sequence length.

正解ラベル	予測ラベル														
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	16	0	0	0	0	0	0	0	0	0	1	3	0	0
2	0	0	18	0	0	0	0	2	0	0	0	0	0	0	0
3	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	1	19	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	20	0	0	0	0	0	0	0	0
7	0	0	1	0	0	0	0	19	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	19	1	0	0
12	0	1	0	0	0	0	0	0	0	0	0	1	18	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20

表 6: 100%の長さを識別した際の混同行列
Table 6 Confusion matrix for 100% sequence length.

正解ラベル	予測ラベル														
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	13	0	0	0	0	0	0	0	0	0	4	3	0	0
2	0	0	18	0	0	0	0	2	0	0	0	0	0	0	0
3	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	1	19	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	20	0	0	0	0	0	0	0	0
7	0	0	1	0	0	0	0	19	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0
11	0	1	0	0	0	0	0	0	0	0	0	15	4	0	0
12	0	2	0	0	0	0	0	0	0	0	0	0	18	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20

けでなく, 引数や戻り値などの詳細な情報を特徴量として活用することや, 注意機構（Attention Mechanism）を導入して, 重要度の低い API コールを無視させる手法を導入することなどを検討している.

参考文献

[1] FFRI セキュリティ: 未知の脅威に対抗する「先読み対策」とは, https://www.ffri.jp/special/special_1(2024-08-09 参照)

[2] 朝倉 紗斗至, 中川 恒, 押場 博光, 吉浦 裕, 市野 将嗣 : 動的解析ログを用いた特徴量の予測によるマルウェアの早期機能推定に関する検討，Computer Security Symposium 2020，2C3-3(2020)

[3] Chen Li1, Junjun Zheng : API Call-Based Malware Classification Using Recurrent Neural Networks, Journal of Cyber Security and Mobility, Vol. 10 3, 617–640(2021)

[4] 大江 弘晃，毛利 公一，鄭 俊俊 : API コール情報を用いた注意機構付き LSTM による マルウェアの特徴抽出と分類，IPSJ SIG Technical Report，Vol.2021-EIP-94 No.10(2021)

[5] 丸若 弘介, 青木 茂樹, 宮本 貴朗 : API コールの時系列情報に注目した LSTM によるマルウェアの早期検知と分類, Computer Security Symposium 2024

[6] Pascal Maniriho, Abdun Naser Mahmood, Mohammad Jabed Morshed Chowdhury : API-MalDetect: Automated malware detection framework for windows based on API calls and deep learning techniques, Journal of Network and Computer Applications Volume 218, September 2023, 103704(2023)

[7] 益留 歩夢, 班 涛, 高橋 健志, 林 宗男, 森川 智博 : 深層学習ベースのスタッキングを用いた Android マルウェア検出, Computer Security Symposium 2024

[8] LOOCV について解説&Python で実装する【機械学習入門 8】, https://datawokagaku.com/loocv/ (2025-08-16 参照)

[9] MWS Datasets, https://www.iwsec.org/mws/2020/files/Soliton_Dataset_2020.pdf (参照 2025-08-16)

[10] MWS Datasets, https://www.iwsec.org/mws/2021/files/Soliton_Dataset_2021.pdf (参照 2025-08-16)

[11] InfoTrace MarkII for Cyber, https://www.soliton.co.jp/mark2/(参照 2025-08-16)