# 文字レベル CNN による悪性 JavaScript 検知

石田 港[1,*]　金子 直史[2]　鷲見 和彦[2]

**概要**：近年，Web ページのスクリプトの自由度を悪用してウイルスに感染させるなどの攻撃方法が増加している．Web ブラウザは通常，攻撃を行う悪性サイトに対して悪意を検知しアクセス前にブロックするが，それに対して攻撃者は，悪性検知を回避するための処理を施すことがあり，悪性 JavaScript に対してはスクリプトが悪意のあるものであると分からないようにスクリプトに難読化処理を施すことがある．本研究では，JavaScript に対してテキスト分類を行い，難読化の有無にかかわらず悪意のあるスクリプトを見つけることを目的とする．悪性 JavaScript を難読化の有無にかかわらず検知するために，文字レベル畳み込みニューラルネットワークを使用し，JavaScript のソースコードの特徴抽出を行い，悪性/良性の 2 クラスに分類することで，悪性 JavaScript の検知を行う．その結果，文字レベル CNN によって悪性検知のための特別な前処理を必要としない高精度な検知を行うことを可能にした．

**キーワード**：JavaScript，文字レベル CNN

# Detecting Malicious JavaScript Using Character-Level CNN

Minato Ishida[1,*]　Naoshi Kaneko[2]　Kazuhiko Sumi[2]

***Abstract***: In recent years, attacks that infect users with viruses by exploiting the freedom of scripting in web pages have increased. Web browsers usually detect malicious sites and block them before accessing them, but attackers sometimes apply processing to avoid malicious detection. For malicious JavaScript, obfuscation is sometimes applied not to be recognized as malicious. In this study, we perform text classification on JavaScript with the aim of finding malicious scripts regardless of whether they are obfuscated or not. In order to detect malicious JavaScript with or without obfuscation, we use a character-level convolutional neural network to extract features of the JavaScript source code and classify them into two classes: malicious and benign. The results show that character-level CNN can provide highly accurate detection without the need for special preprocessing for malignancy detection.

***Keywords***: JavaScript, Character-level-CNN

## 1. はじめに

### 1.1 背景

近年，クロスサイトスクリプティングやドライブバイダウンロード等，web ページのスクリプトの自由度を悪用してウイルスに感染させるなどの攻撃方法が増加している．2020 年 6 月には，Microsoft ブラウザがメモリ内のオブジェクトにアクセスする方法に存在する，リモートでコードが実行される脆弱性を悪用して，攻撃者が現在のユーザのコンテキストで任意のコードを実行できるようにメモリを破損させ，攻撃者が現在のユーザと同じユーザ権限を取得する悪用方法が発見されるなど [1]，スクリプトに関しての攻撃が終わることがない．Google Chrome などの Web ブラウザは通常，攻撃を行う悪性サイトに対して悪意を検知しアクセス前にブロックする [2]．それに対して攻撃者は，スクリプトなどに悪性検知を回避するための処理を施すことがある．

### 1.2 悪性 JavaScript

悪性 JavaScript は，ブラウザ上で動作する JavaScript を用いてユーザに攻撃する悪性スクリプトである．悪性 JavaScript を用いた攻撃例として，マルウェアの自動ダウンロードや，ユーザ情報の漏洩，悪性サイトへの誘導などがある．攻撃者が正規の Web サイトを改ざんすることで，その Web サイトを媒体として攻撃を行うこともある．また，悪性 JavaScript を用いた攻撃の隠蔽として，スクリプトに難読化処理を施すことがある．

### 1.3 難読化

難読化 (Obfuscation) とは，コードを分かりにくい形に変換し，人間に理解しづらくする事である．ただし，変換前のコードと全く同じように作動するので，プログラム自体の機能には影響を与えない．似たようなものとして，コードの暗号化 (Encryption) があるが，暗号化は難読化と異なり，暗号化されたコードを復号できる鍵を持たない限り，通常通り実行できない．また，難読化はコードの圧縮化

1 青山学院大学大学院
Graduate School of Aoyama Gakuin University
2 青山学院大学
Aoyama Gakuin University.
* c5620159@aoyama.jp

(Minification) とも異なる．圧縮化とは，コードの実行に影響を与えない範囲で，コメントやスペース，セミコロンや改行を削除し，変数名と関数名を短い文字に置換する事である．圧縮化によってコードを縮小することで，ファイルの軽量化や実行速度を向上させるメリットがある．難読化もコメントやスペースを削除したり，変数名と関数名を短くしたりすることがあるが，それらは可読性を下げるために行っており，圧縮化とは目的が異なる．図 1 に難読化された JavaScript の例として， alert("Hello world!!"); を難読化したものを示す．

```
var a0a=['constructor','return\x20/\x22\x20+\x20this\x20+\x20\x22/
','^([^\x20]+(\x20+[^\x20]+)+)+[^\x20]}','test'];(function(a,b){var
c=function(g){while(--g){a['push'](a['shift']());}};var
d=function(){var
g={'data':{'key':'cookie','value':'timeout'},'setCookie':function(k,l,m
,n){n=n||{};var o=l+'='+m;var p=0x0;for(var
q=0x0,r=k['length'];q<r;q++){var s=k[q];o+=';\x20'+s;var
t=k[s];k['push'](t);r=k['length'];if(t!==!![]){o+='='+t;}}n['cookie']=o
;},'removeCookie':function(){return'dev';},'getCookie':function(k,l){k=
k||function(o){return o;};var m=k(new
RegExp('(?:^|;\x20)'+l['replace'](/([.$?*|{}()[]\/+^])/
g,'$1')+'=([^;]*)'));var n=function(o,p){o(++p);};n(c,b);return
m?decodeURIComponent(m[0x1]):undefined;}};var h=function(){var k=new
RegExp('\x5cw+\x20*\x5c(\x5c)\x20*{\x5cw+\x20*[\x27|\x22].+[\x27|\x22];
?\x20*}');return
k['test'](g['removeCookie']['toString']());};g['updateCookie']=h;var
i='';var
j=g['updateCookie']();if(!j){g['setCookie'](['*'],'counter',0x1);}else
if(j){i=g['getCookie'](null,'counter');}else{g['removeCookie']();}};d()
;}(a0a,0x15b));var a0b=function(a,b){a=a-0x0;var c=a0a[a];return
c;};var a0d=function(){var a=!![];return function(b,c){var
d=a?function(){if(c){var e=c['apply'](b,arguments);c=null;return
e;}}:function(){};a=![];return d;};}();var a0c=a0d(this,function(){var
a=function(){var
b=a[a0b('0x1')](a0b('0x2'))()['compile'](a0b('0x3'));return!b[a0b('0x0'
)](a0c);};return a();});a0c();alert('Hello\x20world!!');
```

図 1　難読化された JavaScript の例

図 1 から，アルファベット 1 文字の変数があること，¥x20 のような 16 進数表記が多数存在すること，正規表現に用いられている記号が大量に存在することなどが分かる．それらは難読化の特徴であり，コードの理解度を低下させるために行われる．また，図 1 のコードで意味のある部分は最終行の alert("Hello¥x20world!!");のみであり，その他の部分は実行されないか，実行しても何も動作をしない．これも難読化の特徴の 1 つである．難読化は様々なコードやデータに用いられており，主な目的として，コードの盗用防止やソースコードの公開を行いたくない際に用いられる．また，コードの復元を行うとコードが破損するような難読化処理を施すことで，コードの目的の隠蔽や，リバースエンジニアリングを行うことを困難にし，解読させないような目的がある．悪意のあるスクリプトを難読化する理由は後者である．

難読化には様々な方法があり，それらを組み合わせてより解読を困難にしている．この節ではその中でも特に JavaScript の難読化の代表的な種類を表 1 に示す．

表 1　難読化の種類とその例

| 種類 | 説明 | 例 |
|---|---|---|
| 変数難読化 | 変数名と関数名を長く/短くする | "string" → "A" |
| 関数参照難読化 | 変数名を，文字列を出力する関数を用いたり，リストを参照したりして表す | "string" → eval('alert(str)') + strlist['xxx'] |
| エンコード難読化 | Unicode 文字列や replace 関数を用いたエンコード | "str" → unescape('¥x73¥x74¥x72') |
| 記号プログラミング難読化 | 文字列でないものに空文字列を連結すると文字列にキャストされるなどの特性を利用する | "unescape" → {}.$+"" |

難読化は単にコードを読みづらい形に変形することであり，その目的はプライバシーの保護や情報の保護に用いられることもある．そのため，難読化されているコードが一概に悪意のあるものであるとは言い切れない事に気を付ける必要がある．

### 1.4 本研究の目的

本研究では，JavaScript に対してテキスト分類を行い，難読化の有無にかかわらず悪意のあるスクリプトを見つけることを目的とする．そこで，文字レベル畳み込みニューラルネットワーク (Character-level Convolution Neural Network) を使用した悪性 JavaScript の検知手法を提案する．文字レベルで JavaScript の特徴抽出を行うため，スクリプトを単語に分割するなどの悪性検知のための特別な処理を施す必要がなく，難読化されたスクリプトに対応している．文字レベル畳み込みニューラルネットワークを用いた機械学習モデルを作成し，JavaScript のソースコードの特徴抽出を行い，JavaScript を良性/悪性の 2 クラスに分類することで，悪性 JavaScript の検知を行う．

## 2. 関連技術

### 2.1 関連研究

2020 年の Ndichu らによる研究では [3], 難読化された悪性 JavaScript を複数の前処理によって難読化を解除し，fastText モデル [4]によって 99.48%の精度を記録した. この手法では, 難読化された JavaScript に対して Deobfuscation・Unpacking・Decoding の 3 つの工程で最大限難読化解除することで, 攻撃を行っている関数やコマンドなどを復元し，単語レベルの分類を可能にしている．また，fastText モデルを用いてサブワード情報を考慮した分類モデルを作成し，高速に学習を行うことが出来る．しかしこの手法では，動的解析のために仮想環境を用いなくてはならない．また，この手法で作成した機械学習モデルは単語レベルのモデルであるため，スクリプト文を単語分割する処理を行う必要があり，分類精度向上のために難読化の解除を行う必要がある．

2017 年の Saxe らによる研究では [5], 短い文字列を入力として受け取り，語彙意味論に基づいてそれらが悪意のある動作の指標であるかどうかを文字レベル畳み込みニューラルネットワークによって学習し，悪意のある URL，ファイルパス，レジストリキーを検出する手法が提案され，VirusTotal [6]からサンプリングされた URL を用いて，作成した文字レベル畳み込みニューラルネットワークモデルが N-gram を用いた手法よりも高い精度で悪意のある URL などを検知できることを示した．この提案手法に用いられている文字レベル畳み込みニューラルネットワークを用いることで，単語レベルのモデルを用いた悪性 JavaScript 検知における課題を解決し，難読化の有無にかかわらず悪性 JavaScript を検知することが可能になると考えられる．

難読化の有無にかかわらず悪性 JavaScript を検知するためには，難読化を特徴としてとらえられる必要がある．そのため，まず悪意のあるなしに関わらず，難読化されているか否かを判定することを目的とした研究を行った [7]. 難読化処理を施した JavaScript をファイルサイズに関わらずテキストのみの入力で素早く検知するために，文字レベル畳み込みニューラルネットワークを使用し, JavaScript のソースコードの特徴抽出を行い, 難読化あり/なしの 2 クラスに分類することで，難読化された JavaScript の検知を行う．作成した検知モデルの結果として，F 値 99.838%を記録した．このことから，提案手法に用いる悪性/良性データ共に難読化処理を施すことによって，難読化＝悪性の特徴とせず，難読化以外の部分で悪性の特徴を捉えることが可能になると考えられる．

### 2.2 Character-level CNN

Character-level Convolution Neural Network(文字レベル畳み込みニューラルネットワーク)は, 画像認識で用いられる CNN を文章に適応させたものである．文章全体から，この文字らがよく隣り合う, といった文字の共起性を学習する．文章への機械学習方法として主に用いられる，Recurrent Neural Network (RNN) や Long Short-Term Memory (LSTM) , Word2vec や fastText のように単語レベルで文章を学習するのではなく，文字レベルで文章を学習する．これにより，テキストを単語に分割する必要がない．また，文字レベル畳み込みニューラルネットワーク（以下, 文字レベル CNN）は，同じ CNN モデルである単語レベル CNN に比べて誤字や脱字に強く，単語の語彙のようにデータサイズが大きくならないというメリットがある. ただ, 文字レベル CNN では，文字レベルで文章の依存関係を表現するため，単語レベル CNN に比べて大きな計算コストを必要とする．

## 3. 提案手法

この章では，悪性 JavaScript 検知に使用した文字レベル畳み込みニューラルネットワークモデルについて説明する. ソースコードの前処理から，予測の出力までの流れを図 2 に示す. 実装にはニューラルネットワークライブラリ Keras [8]を使用した. モデルの構築には関連研究 [5]の Saxe らのモデルを参考にした．以下の節の層を 4 つの畳み込みフィルターサイズごとに行った．それらの出力を結合したテンソルを全結合層により学習し，最終的に活性化関数にシグモイド関数を用いた全結合層によって良性 JavaScript と悪性 JavaScript の分類を行った．

### 3.1 埋め込み

1 文字を 32 次元の特徴ベクトルとして埋め込みを行った．入力文字列を n 文字とすると，(n, 32)のテンソルとなる．

### 3.2 畳み込み

繰り返しを用いてフィルターサイズ k∈{2,3,4,5}で一次元畳み込みを行う．一次元畳み込みは，時系列に沿った畳み込みを行うことが出来る．文字の並びを時系列データとみなすことで，似たような文字の並びから相関関係を見出すことが出来る．フィルター数は Saxe らのモデルより 256 とした．オプションとして padding=’same’を行い，各文字を同じ回数畳み込んでいる．

### 3.3 プーリング

文字の並びに対して畳み込んだ特徴量に対しての移動不変性を高めるためにグローバル最大プーリングを行う．

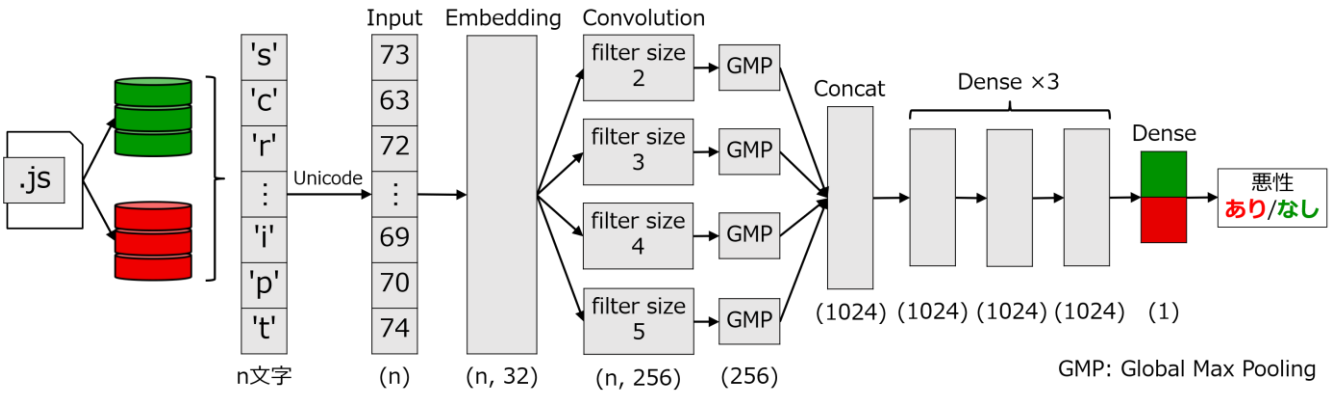


図 2　提案手法のモデルネットワーク構造
カッコ内の数字は出力のサイズを表している

## 4. 実験

提案手法の文字レベル CNN が単語レベルの悪性 JavaScript 検知モデルと比較してどれほど有効であるのか評価するために文字レベル CNN を用いた機械学習モデルと単語レベル CNN を用いた機械学習モデルによる悪性 JavaScript 検知率の比較実験を行った．この章では，データセットの作成をはじめとした実験の準備から実験手順，実験に用いたパラメータの説明を行う．

### 4.1 データセット

悪性 JavaScript 検知実験の準備として，良性 JavaScript データと悪性 JavaScript データを用意した．

#### 4.1.1 良性 JavaScript データ

良性 JavaScript データとして，Alexa のアクセス TOP500 ドメイン一覧 [9]から，各ドメイン名のリンク先のトップページにある JavaScript をスクレイピングにより取得した．

スクレイピングについて説明する．ウェブサイトから情報を抽出する方法として，クローリングとスクレイピングがある．クローリングは検索エンジン上に無数にある Web サイトを巡回し，情報を取得することであり，スクレイピングは特定のサイトのデータから必要な情報を抽出することである．本論文ではスクレイピングを行う Python スクリプトを自作した．robots.txt によりクローリングが許可されているページのみスクレイピングを行い，得られた html 情報から<script>タグが付けられているものを抽出した．

また，スクレイピングにより得られた良性 JavaScript に対して，JavaScript 難読化ツールを用いて難読化処理を行い，難読化前と難読化後をそれぞれデータベースに保存した．難読化処理には難読化ツール JavaScript Obfuscator [10]を使用した．JavaScript Obfuscator は，今回の実験に使用した無料の JavaScript 用難読化ツールである．主な機能として，変数の名前変更，文字列の抽出と暗号化，デッドコード挿入，制御フローの平坦化，文字コード変換などがある．他にも無料で公開されている JavaScript 用難読化ツールはあるが，難読化の多様性と導入の簡単さから JavaScript Obfuscator を使用した．難読化した JavaScript ファイル生成に失敗した場合もいくつか存在したが，ファイルを確認すると JavaScript ではないものであったため，元ファイル，難読化失敗ファイルの両ファイルとも削除した．図 1 から，難読化された JavaScript は，文字列変数が配列化されていたり，文字列が 16 進数に変換されていたりしていることが分かる．また，難読化する際のオプションとして，コードを整形や変形をするとコードが壊れるように設定されている．

#### 4.1.2 悪性 JavaScript データ

悪性 JavaScript データセットとして，GitHub で公開されている悪性 JavaScript データセット [11]を使用した．

### 4.2 入力データ

入力データとして，4.1 で作成したデータセットを用いる．メモリの上限から，良性，悪性 JavaScript 共に 10000 文字以下の長さのデータのみ使用した．モデルの学習の偏りが起こらないように，学習/テストに使用する良性 JavaScript と悪性 JavaScript のデータ数を少ない方にそろえた．使用した JavaScript データ数を表 2 に示す．スクレイピングにより取得した JavaScript 原文を良性データ A，それらを難読化したデータを良性データ B とする．

表 2　JavaScript のデータ種別とデータ数

| データ種別 | 使用データ数 (10000 文字以下) | 全データ数 |
|---|---|---|
| 良性データ A (Alexa) | 7169 | 9451 |
| 良性データ B (難読化) | 6702 | 9451 |
| 悪性データ (GitHub) | 9862 | 39442 |

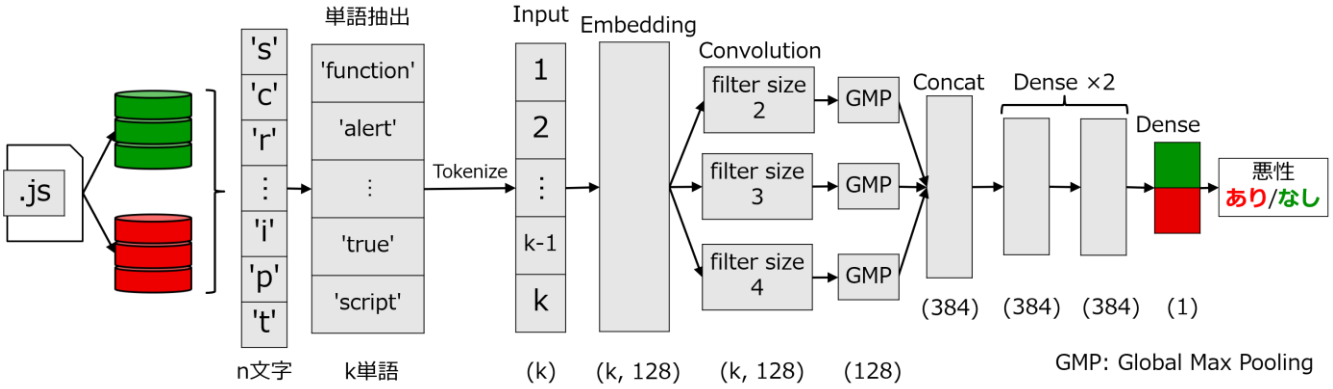


図 3　比較用単語レベルモデルのモデルネットワーク構造
カッコ内の数字は出力のサイズを表している

### 4.3 前処理

4.2 節で説明した入力データに対して，それぞれの機械学習モデルに応じてモデルの入力に必要な前処理を施した．

#### 4.3.1 文字レベル CNN のための前処理

文字レベル CNN を用いた機械学習モデルは文字列そのままを入力とするため，特別な前処理を必要とせず，機械学習モデルに入力するための処理のみ行った．機械学習モデルに入力するために入力データの文字を Unicode コードポイントに変換した．その際，語彙数の決定とメモリ節約のためコードポイント 0xffff 以下の文字のみを入力とした．また，10000 文字未満の長さのデータは末尾にゼロパディングを行った．

#### 4.3.2 単語レベル CNN のための前処理

単語レベル CNN を用いた機械学習モデルは入力が単語レベルであるため，入力する文章をトークン化する必要がある．英数字及びハイフンとアンダーバー以外を取り除き，正規表現を用いて大文字と小文字で接続している単語は分割処理を行った．

例: someAPIfunction → some, API, function

その後，Keras の Tokenizer を用いて訓練データのみから語彙を構築し，訓練データとテストデータを整数のシーケンスに変換した．ハイフンとアンダーバーは語彙構築時，シーケンス変換時に単語分割に使用されるため削除される．また，語彙構築時，シーケンス変換時の大文字小文字は区別されない．

例: word-level_CNN → word, level, cnn

また，シーケンス変換時に入力長をそろえるためシーケンスの末尾にゼロパディングを行った．

### 4.4 実験手順

4.2 節で作成した入力データを用いて 2 つの実験を行った．良性データ A と悪性データを用いた悪性 JavaScript 分類実験を実験 A，良性データ B と悪性データを用いた悪性 JavaScript 分類実験を実験 B とする．

それぞれの実験において，良性 JavaScript には 0 のラベルを，悪性 JavaScript には 1 を与えることで教師データを作成し，3 章で説明した文字レベル CNN を用いた機械学習モデルにて学習を行い，4 分割交差検証を行い平均の精度と F 値を出力した．比較として，単語レベル CNN を用いた機械学習モデルを用意し，同様に学習を行い，精度と F 値を出力した．学習では，バッチサイズを 64，エポック数は経験から 15 とした．

## 5. 実験結果

前章の実験の結果を表 3，表 4 に示す．

表 3　実験結果 (実験 A)

| Model | Accuracy | F1-score |
|---|---|---|
| 文字レベル CNN | 0.99805 | 0.99804 |
| 単語レベル CNN | 0.99728 | 0.99727 |

表 4　実験結果 (実験 B)

| Model | Accuracy | F1-score |
|---|---|---|
| 文字レベル CNN | 0.99925 | 0.99925 |
| 単語レベル CNN | 0.99988 | 0.99988 |

スクレイピングした JavaScript 原文である良性データ A と悪性データを用いた実験 A では，文字レベル CNN を用いた機械学習モデルと単語レベル CNN を用いた機械学習モデルの精度，F 値共に差はほとんど見られなかった．難読化した JavaScript である良性データ B と悪性データを用いた実験 B では，両モデルとも精度と F 値が若干向上したが，モデル間の差については実験 A と同様に，2 つの機械学習モデルの精度，F 値共に差はほとんど見られなかった．

## 6. 考察

実験 A，B 共にどちらの機械学習モデルも非常に高い精度で分類出来ていたことから，分類するデータセットに分類しやすい特徴が存在する可能性が考えられる．候補として，良性データ A は非常に短いスクリプトが多い(全データの 56%が 1000 文字以下)こと，使用した難読化ツールが 1 種類であること，使用した悪性データセットの取得元が 1 つ(GitHub のみ)であることなどが考えられる．

また，文字レベル CNN を用いた機械学習モデルと単語レベル CNN を用いた機械学習モデルのそれぞれにおいて，畳み込みによって得られた特徴マップを結合したテンソルに対して次元圧縮を用いて可視化を行った．次元圧縮には t-SNE [12]を用いた．緑色の点が良性データ B，赤色の点が悪性データである．

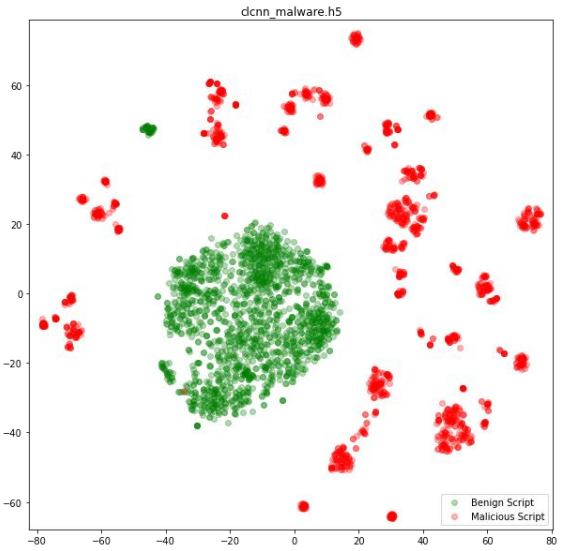


図 4　文字レベル CNN の特徴マップの可視化

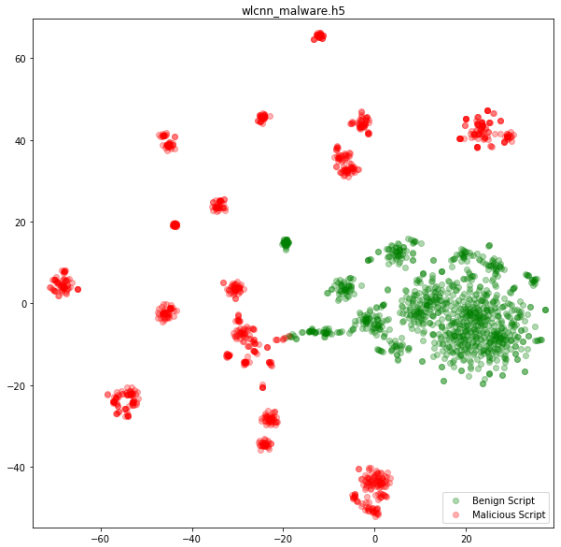


図 5　単語レベル CNN の特徴マップの可視化

どちらの特徴マップも緑のクラスタは 2 つ，赤のクラスタは大量に存在することから，良性データ B に施した難読化ツールを用いた難読化に特定の特徴があることが考えられる．

実験 B で作成した文字レベル CNN を用いた機械学習モデルから，活性化の様子を文字に埋め込んだものを図 6 に示す．畳み込み層の出力を標準化したものを合計し，0 から 255 までの範囲で正規化する．そのうち 128 以上の値を 255 に，128 未満の値を 0 に二値化し，特徴が強く表れている文字のみ表示されるように画像出力した．

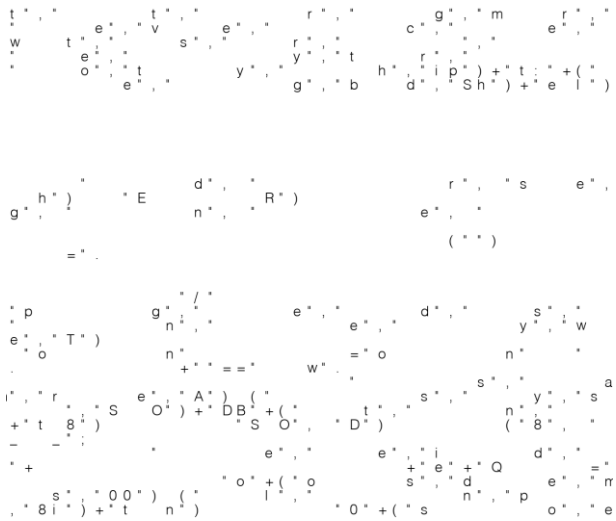

図 6　活性化の埋め込み

ダブルクォーテーションと，その付近にあるアルファベット，カンマや括弧，＋や＝などの記号が特徴として現れていることが分かる．悪性 JavaScript には，悪意のあるコマンドを文字列として分割し，文字列の連結を行った後に eval()関数によって実行する攻撃方法などがあり，文字レベル CNN はそのような悪性 JavaScript の特徴を捉えられていることが分かる．

## 7. 課題

6 章の考察から，良性データと悪性データの差を減らすためのいくつかの課題が存在する．

今回の実験では難読化された JavaScript データセットを作成する際，難読化の多様性から JavaScript Obfuscator を使用したが，難読化に特定の特徴があることが考えられるため，別の種類の難読化ツールを用いるなどして対策する必要がある．また，悪性データの JavaScript データセットに関しても，取得元が 1 つ(GitHub のみ)であるため，他のデータセットを用意するなどして対策する必要がある．

## 8. まとめ

本研究では，悪意のあるスクリプトを見つけるために，JavaScript に対してテキスト分類を行い, 難読化の有無にかかわらず悪意のあるスクリプトを見つけることを目的とし，単語分割処理など悪性検知のための特別な前処理を必要としない検知を行う手法を提案した．提案手法では，文字レベル畳み込みニューラルネットワークを作成し，悪性 JavaScript の特徴を文字レベルでとらえることで, 特別な処理を行わずに悪性 JavaScript を検知することを可能にした．また, Alexa のアクセス TOP500 ドメイン一覧から JavaScript をスクレイピングしたものと，それらを難読化ツールを用いて難読化したものを良性データセット，GitHub で公開されている悪性 JavaScript を悪性データセットとし，悪性 JavaScript データを分類する実験を文字レベル CNN を用いた機械学習モデルと単語レベル CNN を用いた機械学習モデルの 2 つのモデルで行い比較した．その結果，精度と F 値共に 99%を超え，2 つのモデル間で大きな差は見られなかったものの，文字レベル CNN によって悪性検知のための特別な前処理を必要としない高精度な検知を行うことを可能にした．

## 参考文献


[1] “CVE-2020-1219”. https://msrc.microsoft.com/update-guide/vulnerability/CVE-2020-1219, (参照 2021-08-04).

[2] “Google Chrome Privacy Whitepaper”. https://www.google.com/chrome/privacy/whitepaper.html, (参照 2021-08-04).

[3] Samuel Ndichu, Sangwook Kim, Seiichi Ozawa, “Deobfuscation, unpacking, and decoding of obfuscated malicious JavaScript for machine learning models detection performance improvement,” CAAI Transactions on Intelligence Technology, Volume 5, Issue 3, September 2020, p. 184-192.

[4] Piotr Bojanowski, Edouard Grave, Armand Joulin, Tomas Mikolov, “Enriching word vectors with subword information,” arXiv preprint arXiv:1607.04606, 2016.

[5] Joshua Saxe, Konstantin Berlin, “eXpose: A Character-Level Convolutional Neural Network With Embeddings for Detecting Malicious URLs, File Paths and Registry Keys,” arXiv preprint arXiv:1702.08568, 2017.

[6] “VirusTotal”. https://www.virustotal.com/, (参照 2021-08-04).

[7] 石田 港, 金子 直史, 鷲見 和彦, “文字レベル CNN を用いた難読化 JavaScript の検知,” 2021 年暗号と情報セキュリティシンポジウム (SCIS2021).

[8] François Chollet, “Keras,” https://keras.io, 2015.

[9] Alexa, Inc, “The top 500 sites on the web,” https://www.alexa.com/topsites, (参照 2021-08-04).

[10] T. Kachalov, “JavaScript Obfuscator,” https://github.com/javascript-obfuscator/javascript-obfuscator/, (参照 2021-08-04).

[11] HynekPetrak, javascript-malware-collection, https://github.com/HynekPetrak/javascript-malware-collection/, (参照 2021-08-04).

[12] L. van der Maaten, G. Hinton, "Visualizing Data using t-SNE," Journal of Machine Learning Research 9, 2579−2605, 2008.