Vol.2014-DPS-158 No.21
Vol.2014-CSEC-64 No.21
2014/3/6

文字出現頻度をパラメータとした機械学習による悪質な難読化 JavaScript の検出

西田雅太[†1a)] 星澤裕二[†1b)] 笠間貴弘[†2c)]
衛藤将史[†2d)] 井上大介[†2e)] 中尾康二[†2f)]

近年増加しているドライブバイダウンロード攻撃では，JavaScript を介して攻撃を行うものがあり，悪意のある JavaScript を検出する手法が希求されている．本稿では，難読化が施された JavaScript の文字出現頻度が一般の JavaScript とは異なる傾向があることに着目し，スクリプトの文字出現頻度を機械学習のパラメータとすることで，悪意のある難読化スクリプトを検出する手法を提案する．また提案手法の検証として，一般サイトの JavaScript と MWS データセット内の D3M 攻撃通信データの JavaScript を入力として学習した結果を示す．

Obfuscated Malicious JavaScript Detection using Machine Learning with Character Frequency

MASATA NISHIDA[†1a)] YUJI HOSHIZAWA[†1b)] TAKAHIRO KASAMA[†2c)]
MASASHI ETOU[†2d)] DAISUKE INOUE[†2e)] KOJI NAKAO[†2f)]

Today the number of Drive-by-Download attacks using JavaScript has increased. Therefore we need an efficient method to detect malicious JavaScript. In this paper, we focus our attention on a bias of character frequency of obfuscated malicious JavaScript. We will propose the use of machine learning with character frequency to detect obfuscated malicious JavaScript. This paper will also evaluate the proposed method by using various JavaScript in benign web sites and D3M pcap of MWS dataset.

1. はじめに

近年，ブラウザを利用したドライブバイダウンロードによる攻撃が増加している．ドライブバイダウンロード攻撃では，JavaScript によって攻撃が行われることがあり，悪意のある JavaScript を検出する手法が希求されている．

ドライブバイダウンロード攻撃に用いられる JavaScript には，スクリプトの挙動が簡単にわからないよう難読化が施されているものがある．

ドライブバイダウンロード攻撃の JavaScript に用いられる難読化は，同一の攻撃では同じようなアルゴリズムで難読化されていることがあり，こうした難読化手法自体が悪意のある JavaScript の特徴点として捉えることが出来る．

本稿では，JavaScript 内の文字出現頻度が悪質な JavaScript の難読化アルゴリズムの特徴を表すと仮定し，JavaScript 内の文字出現頻度を機械学習のパラメータとすることで，悪意のある難読化スクリプトを検出する手法を提案する．

また，提案手法を検証するために一般サイトの JavaScript と MWS Dataset 2013[1]内の D3M 攻撃通信データの JavaScript を検証用のデータセットとして検証を行う．

検証では，一般サイトと D3M 攻撃通信データの JavaScript において JavaScript 内の文字出現頻度の傾向に差異があることを示した上で，文字出現頻度を特徴パラメータとしてサポートベクターマシン（以降，SVM）を用いて学習した結果を示す．

†1 株式会社セキュアブレイン 先端技術研究所
Advanced Research Laboratory, SecureBrain Corporation
†2 独立行政法人 情報通信研究機構
National Institute of Information and Communications Technology
a masata_nishida@securebrain.co.jp
b yuji_hoshizawa@securebrain.co.jp
c kasama@nict.go.jp
d eto@nict.go.jp
e dai@nict.go.jp
f ko-nakao@nict.go.jp

2. 提案手法

2.1 難読化 JavaScript

文字列の置き換えなどにより第三者によるソースコードの解析を困難にする技術を難読化という．JavaScript はサーバからソースコードをダウンロードし，クライアントサイドで実行されるため，アルゴリズムの隠蔽などを目的として一般の JavaScript においても難読化が施されている場合がある．一方で，ドライブバイダウンロード攻撃に用いられる JavaScript においても難読化が施されていることがある．難読化された JavaScript は，それぞれが持つ難読化を解除するアルゴリズムにより，スクリプトの変換を行い，eval()メソッドなどにより実行される．

難読化された悪質な JavaScript の例として，2009 年に観測された Gumblar の JavaScript を以下の図 1 と図 2 に例示する．

Vol.2014-DPS-158 No.21
Vol.2014-CSEC-64 No.21
2014/3/6

```
document.write(unescape('%3CrNssycvtrRZiRZpt1Jc%
2FscV14riiHJpt%3E').replace(
    /rN|RZ|sy|V14|iHJ|vt|1Jc/g,""));
```

図 1 Gumblar の JavaScript (1)
Fig. 1 Gumbler's JavaScript (1)

図 1 のスクリプトでは，String.replace()メソッドと unescape()メソッドを用いて<script>タグを含んだ文字列を生成し，document.write()メソッドによって<script>タグを HTML コンテンツに追加する．<script>タグには，外部サイトの JavaScript の URL が src 属性に指定されており，ブラウザは外部の JavaScript を新たに読み込む．

```
(function(PI4i){
    var x9P5r=('-76a-72-20a-3d-22Script-45ngine-
        22-2cb-3d-22V-65rsio-6e-28)+-22-2cj-3d-
        22-22-2cu-3dna-76igato-72-2euser-41gen-
        22+j+-22-3e-3c-5c-2fsc-72i-70-74-3e-22)-
        3b-7d').replace(PI4i,'%');
    var GXf9Y=unescape(x9P5r);
    eval(GXf9Y)})(/-/g);
```

図 2 Gumblar の JavaScript (2)
Fig. 2 Gumblar's JavaScript (2)

図 2 のスクリプトでも，String.replace()メソッドと unescape()メソッドを組み合わせて使用し，文字列を変換することによって実行すべき JavaScript を得る．その上で eval()メソッドにより JavaScript の実行を行う．

本スクリプトも最終的には document.write()メソッドにより外部の JavaScript を読み込む<script>タグが生成される．

このように，Gumblar の JavaScript は，複数のアルゴリズムで難読化の解除と実行が行われることが確認されているが，同一のアルゴリズムで難読化を解除する場合においても，置換に用いられる文字列を変更するなどによって改変されたスクリプトが多数観測されている．

2.2 難読化スクリプトの特徴点

前節で示したとおり，難読化された JavaScript は，文字列の変換処理などを行うことにより，難読化を解除した上で本来のスクリプトを実行する．

図 2 の例では，"-（ハイフン）"の記号が多く含まれており，これは String.replace()メソッドにより"%（パーセント）"に置換される．また"-（ハイフン）"の後ろ 2 文字は 16 進文字列（[0-9a-f]）で記述されており，unescape()メソッドによって別の文字列に置換される．その結果，難読化が施されていない JavaScript と比較して，"-（ハイフン）"，数字および一部のアルファベット文字の出現頻度が高くなっている．

このように，難読化された JavaScript は難読化が施されていない JavaScript と比較して，その難読化手法によっては文字の出現頻度に異なる傾向になることが推測できる．

以上のことから，本稿では JavaScript 内の各文字の出現頻度をそのスクリプトの特徴点として難読化された悪質な JavaScript の検知に用いる．

2.3 機械学習による難読化 JavaScript の検出

2.3.1 SVM による機械学習

SVM は，教師あり学習によるパターン認識の一種である．入力された各データに対する距離が最大になるようなマージン最大化超平面を算出することで 2 値分類を行う．

本稿では，JavaScript 内の文字出現頻度を SVM 入力ベクトルとする．JavaScript 内の ASCII 文字のうち制御文字およびスペース（0x20）を除く文字コードが 0x21～0x7e までの計 94 種類の文字を対象とする．この際，文字iの JavaScript 内における出現数をm_iとするとき，JavaScript 内の対象文字数$\boldsymbol{N}$と文字出現頻度$\boldsymbol{F}(i)$を以下のように定義する．

$$\boldsymbol{N} = \sum_i m_i \qquad (1)$$

$$\boldsymbol{F}(i) = \frac{m_i}{\boldsymbol{N}} \qquad (2)$$

$\boldsymbol{F}(i)$は$0.0 \leq \boldsymbol{F}(i) \leq 1.0$の範囲の値をとる．この$\boldsymbol{F}(i)$を SVM の入力ベクトルとする．

なお，本稿では対象とする JavaScript の文字コードを UTF-8 と想定しているため，上記の定義ではマルチバイト文字は無視される．

SVM の学習には，一般の JavaScript と悪質な JavaScript を教師データとし，悪質な JavaScript を検知する SVM モデルの算出を行う．

2.3.2 文字出現頻度を利用するメリット・デメリット

提案手法によって悪質な難読化 JavaScript を検知する際のメリットとして以下の様なものが挙げられる．

- パラメータ計算が単純で高速

提案手法は文字の出現頻度のみを特徴パラメータとするため, パラメータの計算にかかるコストが小さいといえる.

- 不完全な JavaScript も検知対象に出来る

悪質な JavaScript の検知手法として動的に JavaScript をエミュレーションする手法[2]などが提案されているが，そのような手法の場合，検査対象の JavaScript は動的実行環境の JavaScript Interpreter が誤りなく解釈できるものである必要がある．提案手法では，JavaScript の構文解析は行わないため，文法的に誤りのある，またはデータが途中で切れてしまっているような JavaScript でも検知対象とすることが出来る.

また，提案手法のデメリットとしては以下の様なものが考えられる.

- 難読化されていないものは検知が困難

提案手法では難読化によって文字の出現頻度にその JavaScript の固有の特徴点がみられると仮定しており, 難読化されていない JavaScript の場合検知が困難であると考えられる.

- 検査対象の JavaScript のファイルサイズの影響

検査対象の JavaScript がごく小さいサイズの場合，その JavaScript の特徴を表現出来るほどの文字数がなく, 文字出現頻度による特徴抽出が困難になることが懸念される.

- 難読化の解除用のデータが DOM 内に埋め込まれている場合には検知が困難

JavaScript の難読化の手法の一つとして, 難読化解除のためのデータが DOM の中にテキストとして埋め込まれているものが存在する．例えば， タグなどに特定の id 属性を指定した上で, document.getElementById()メソッドにより当該要素を取得し，innerText 属性のテキストを難読化解除に用いることがある.

こうした難読化手法では, JavaScript 自体には変換アルゴリズムしか含まれておらず，文字頻度の特徴が JavaScript 内ではなく DOM の中の要素にあらわれてしまうと考えられる，そのため，提案手法が有効ではないと考えられる.

3. 検証

提案手法の妥当性を検証するために，まず検証用データセットを作成する．その上で，作成した検証用データを用いて検証を行う．検証は以下の 2 点について行う.

- 一般サイトと悪質な JavaScript の間で文字出現頻度（$\boldsymbol{F}(i)$）に差異が認められるか
- 検証用データセットを用いて SVM で分類を行う

3.1 検証用データセットの作成

次項以降の手順で一般のサイトの JavaScript と悪質な難読化 JavaScript を収集し，検証用のデータセットを作成した.

3.1.1 一般サイトの JavaScript

一般サイトの JavaScript を表すデータとして，Alexa[1]が公開しているアクセストップドメインリストのうち，上位 500 位のサイトのトップページにアクセスし，トップページ内に<script>タグで埋め込まれている JavaScript を収集した．<script>タグに src 属性が記述されている場合には，当該の URL にもアクセスし，スクリプトを収集した.

なお，2.3.2 で示したとおり，スクリプトのサイズが小さい場合には，文字出現頻度に特徴がうまく反映されないことが予想されるため 1KB 未満のスクリプトは除外した.

収集した一般サイトの JavaScript についての情報を表 1 に示す.

表 1 収集した一般サイトの JavaScript データ
Table 1 Benign site's JavaScript data

Collected date	2013/11/18
Num of scripts	2,786
Total data size	98,396,453 bytes
Avg. size	35,318 bytes
Max size	2,574,288 bytes
Min size	1,026 bytes

以降，このデータのことを検証用データセット内の良性 JavaScript（Benign JavaScript）という.

3.1.2 悪質な難読化 JavaScript

MWS Dataset 2013[1]の D3M 攻撃通信データのうち, 2011 年から，2013 年の 3 年分のデータを検証用データセットの対象とした. これらの攻撃通信データに含まれる JavaScript のうち，本稿では次の条件に当てはまるものを悪質な難読化 JavaScript として扱う.

D3M 攻撃通信データに含まれる JavaScript のうち，明らかに難読化しているとわかるもののみを検証用データセットでは扱うこととした．また，jQuery など一般的なライブラリと思われるものについてはデータセットから除外した. さらに，3.1.1 と同様スクリプトのサイズが 1KB 未満のものは除外した.

Vol.2014-DPS-158 No.21
Vol.2014-CSEC-64 No.21
2014/3/6

なお，本稿では対象の JavaScript が実際に悪質な挙動を示すのかどうかについては検証を行っていない．上記の条件に合致する D3M 攻撃通信データ内の JavaScript はすべて悪質な難読化 JavaScript として扱うことにした．

悪質な難読化 JavaScript としたデータの情報を表 2 に示す．

表 2 悪質な難読化 JavaScript データ
Table 2 Obfuscated malicious JavaScript data

Collected date	2011/2/8～2013/2/26
Num of scripts	330
Total data size	5,813,480 bytes
Avg. size	17,617 bytes
Max size	324,523 bytes
Min size	1,085 bytes

以降，このデータのことを検証データセット内の悪質 JavaScript（Malicious JavaScript）という．

3.2 JavaScript の文字出現頻度の傾向

検証用データセットの良性 JavaScript と悪性 JavaScript における文字出現頻度の差異を確認するために，両者の文字出現頻度の傾向を分析，比較する．

なお，2.3.1 で定義した 94 種の文字の出現頻度を直感的に比較することは困難なことから，比較をしやすくするために表 3 の通り文字種別を数字，アルファベット，記号の 3 つ分けて定義する．

表 3 文字種別の定義
Table 3 Character type definition

Type	**Definition**
Numbers	[0..9]
Alphabets	[a..zA..Z]
Symbols	0x21～0x7e && Not Numbers && Not Alphabets

表 3 の定義に基づき，検証用データセットにおける文字出現頻度の傾向を分析する．

3.2.1 コンテンツごとの文字出現頻度の分布

検証用データセットのコンテンツごとの文字出現頻度の分布を示す．図 3，図 4 はそれぞれ良性 JavaScript と悪性 JavaScript のコンテンツごとの文字出現頻度の分布を表す．

図 3 と図 4 のグラフでは，横軸はコンテンツごとに文字種別（数字，アルファベット，記号）が占める頻度を表し，縦軸はコンテンツ数を表す．

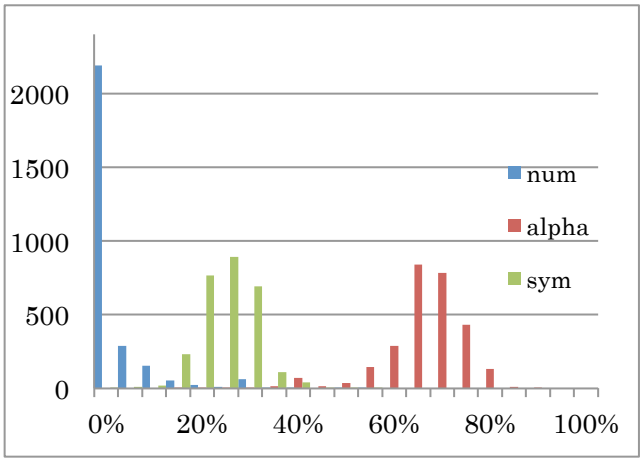

図 3 良性 JavaScript 文字出現頻度の分布
Fig. 3 Character frequency distributions of benign JavaScript

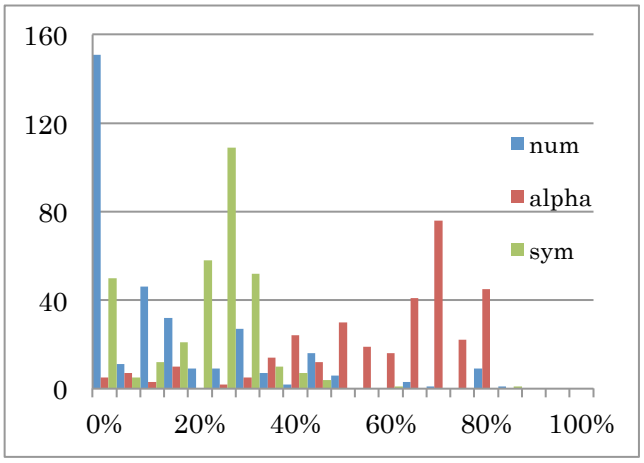

図 4 悪性 JavaScript の文字出現頻度分布
Fig. 4 Character frequency distributions of malicious JavaScript

良性 JavaScript では，コンテンツごとの文字出現頻度はアルファベットと記号はそれぞれ正規分布的な分布を示している，数字は 0%付近で最大値を示しているが，頻度が高くなるにつれてコンテンツ数は減少する．

図 4 に示した悪性 JavaScript では，図 3 の良性 JavaScript と比較して分布にばらつきがみられ，各文字種別において広い範囲の文字出現頻度でコンテンツが分布していることが分かる．

次に，コンテンツごとの文字出現頻度分布の平均，標準偏差などの情報を表 4，表 5 に示す．

表 4 と表 5 の標準偏差の比較からも悪性 JavaScript の方が，ばらつきが大きいことが分かる．

さらに各文字種別で文字出現頻度の最大，最小の値の幅が悪性 JavaScript の方が大きい．特に悪性 JavaScript における数字，記号の文字出現頻度の最大値は 85%以上を示しており，良性 JavaScript の最大値と比較して大きな値である

といえる．

これらの比較から，良性 JavaScript と比較して文字出現頻度が大きく偏っているコンテンツが悪性 JavaScript の中に存在しているといえる．

表 4 良性 JavaScript の文字出現頻度分布の概要
Table 4 Abstract of Character frequency distributions of benign JavaScript

	Numbers	Alphabets	Symbols
Max	56.63%	94.89%	58.38%
Min	0.00%	17.98%	3.00%
Average	4.20%	68.73%	27.07%
Median	1.74%	69.67%	27.00%
Std. Deviation	0.065	0.087	0.059

表 5 悪性 JavaScript の文字出現頻度分布の概要
Table 5 Abstract of Character frequency distributions of malicious JavaScript

	Numbers	Alphabets	Symbols
Max	88.25%	83.67%	85.49%
Min	0.13%	0.00%	0.03%
Average	16.47%	60.43%	23.10%
Median	10.91%	67.72%	25.92%
Std. Deviation	0.882	0.837	0.855

このように，数字，文字，記号という 3 通りの分類においても，良性 JavaScript と悪性 JavaScript の間で文字出現頻度の傾向に顕著な差異が認められた．

3.2.2 数字の出現頻度の多いスクリプト

前節で示した悪性 JavaScript の中で数字の文字出現頻度が 88.25%と最も高かった JavaScript について分析する．

当該 JavaScript の概要を表 6 に示し，JavaScript の一部を図 5 に示す．

表 6 最も数字を含む JavaScript 概要
Table 6 Abstract of top JavaScript includes Numbers

Total file size	14,463
Target data size	14,414
Numbers	12,720 (88.25%)
Alphabets	1,652 (11.46%)
Symbols	42 (0.29%)

この JavaScript では，図 5 に示したように 16 進文字列（[0-9a-f]）で構成された文字列 Vg を 2 文字ずつ分割して”%”を付与し，unescape()メソッドで変換し，JavaScript を含んだ HTML を document.write()メソッドで出力する．難読化解除のもととなる文字列 Vg は約 14KB の長さがあり，これが数字の出現頻度を 80%以上にする要因となっている．

```
var Vg='a06d04937ccdc754e9ebc1c93e37da1309ac
8e3c68746d6c3e...297b0a090909726573202b3d207
368656c6c636f64652e7375627374720909692b2b3b0
a097d0a0a09696620287...';
var HJN = '';
var q = Vg.slice ( 38, 14236 );
for ( K = 38 ; K < 14236 ; K += 2 ) {
  HJN += '%' + Vg.slice ( K, K + 2 );
}
document.write(unescape(HJN));
```

図 5 最も数字を多く含む JavaScript （一部）
Fig. 5 Sample of top JavaScript includes Numbers (partial)

難読化が解除された HTML に埋め込まれている JavaScript は，CVE-2006-0003 の脆弱性をつく攻撃が含まれていた．CVE-2006-0003 は MDAC (Microsoft Data Access Components)に関する脆弱性で，脆弱性が公開されたのは 2006 年であるが，その後長い期間に渡って Exploit Kit などで使用されている脆弱性である．

3.3 SVM による学習

本稿では，SVM を扱うためのライブラリである libsvm[4]を使って SVM による学習を行った．

3.3.1 教師データの作成

2.3.1 で示した文字出現頻度$\boldsymbol{F}(i)$を SVM への入力ベクトルとする．入力は 94 次元となる．

検証用データセットの良性 JavaScript のデータに対し+1 のラベルを，悪性 JavaScript のデータに対しては-1 のラベルを付与し，それぞれのコンテンツごとに算出した$\boldsymbol{F}(i)$と合わせて教師データを作成した．

libsvm には，教師データを最適化するための正規化ツール svm-scale が付属しているが，以下の理由により入力データの正規化は行わず，$\boldsymbol{F}(i)$をそのまま入力データとした．

svm-scale は，デフォルトの設定では各入力次元の最大値最小値をもとにして-1.0 から 1.0 の間に値を正規化する．学習結果に基づいて JavaScript の良悪性を判定する検知システムを構築する場合，教師データのデータ範囲で正規化を行うことは望ましくない．また，$\boldsymbol{F}(i)$は$0 \leq \boldsymbol{F}(i) < 1.0$の範囲の値をとるため，正規化を行わなくても値が妥当な範囲にあると判断した．

3.3.2 カーネルメソッドとパラメータ最適化

学習には SVM のカーネルメソッドとして libsvm のデフ

ォルト設定である RBF（ガウス）カーネルを用いた．このとき，学習時にCとγの 2 つパラメータを適切に設定する必要がある．

本検証では libsvm に付属している grid.py を用いて，グリッドサーチにより網羅的に最適なCとγの探索を行った．この際，交差検定の分割数は 5 とした．

3.3.3 学習結果

3.3.1 で作成した教師データを用いてグリッドサーチを行った結果，$C = 25.22$，$\gamma = 55.72$のとき最も高い正答率（Accuracy）が得られた．

このときの正答率, 悪性 JavaScript の適合率（Precision），悪性 JavaScript の再現率（Recall）を以下の表 7 に示す．

表 7 SVM の学習結果
Table 7 Performance measures of SVM

	Result	**Formula**
Accuracy	98.84%	$\frac{TP+TN}{TP+FP+TN+FN}$
Precision of malicious JavaScript	97.72%	$\frac{TP}{TP+FP}$
Recall of malicious JavaScript	94.35%	$\frac{TP}{TP+FN}$

4. 考察

検証より，検証用データセットの良性 JavaScript と悪性 JavaScript の間には文字の出現頻度の傾向に明らかな差異が見られた．また，悪性 JavaScript の方が文字出現頻度の分布により大きなばらつきが見られた．このことから，良性 JavaScript とは異なった傾向の文字出現頻度で構成された JavaScript が悪性 JavaScript の中に存在しているといえ，文字出現頻度によって悪性 JavaScript の特徴点を表現出来るケースがあることを示している．

SVM による学習を行った結果，正答率（Accuracy）が 98.84%と概ね良好な結果を得ることが出来た．

ただし，本稿では，D3M 攻撃通信データ内の JavaScript について，条件に合致するものは良悪性判定を行わずに検証用データセットとして採用しているため, 悪性 JavaScript のデータセットがどの程度多様な攻撃を含んでいるかといった点については検証が出来ていない．

学習結果が良好だったことも検証用データセットに起因するものであることが考えられる．

5. 関連研究

神薗らは，悪質な JavaScript を検知する手法として，擬似的な DOM 環境を構築し JavaScript をエミュレーションする手法[2]や，JavaScript の抽象構文木を特徴点として扱う手法[5]を提案している．文献[2]では，擬似環境上で JavaScript を動作させることによって，JavaScript の挙動を把握するシステムのため，擬似 DOM 環境が実行する JavaScript に適合している必要があった．また，文献[5]においても, JavaScript のパーサによる構文解析が成功する必要があり，入力となる JavaScript が構文的に正しいものであるという前提のもとに成り立っている．本稿の提案手法では, JavaScript の文字の出現頻度を特徴点としているため, 擬似 DOM 環境の構築や構文解析を必要とせず，これらの研究において問題となっていた制約事項を回避して検知が行える．

また，難読化 JavaScript の特徴点を捉えて機械学習によって検知する試みは，文献[6]で研究が行われている．この研究では，難読化した JavaScript の検知する手法として複数の分類手法を使用している．分類手法としてナイーブベイズや SVM など 4 手法を用いて分類した結果を手法ごとに比較を行っている．分類の際の特徴点としては，JavaScript のサイズ，行数，文字列数などを用いている．

本稿の提案手法と同じ SVM における実験結果では，適合率 92.0%，再現率 74.2%という結果が示されている．

使用しているデータセットが違うため単純比較を行うことは出来ないが，本稿では，文献[6]と同等以上の認識結果が得られた．また，文献[6]でも特徴点の算出に JavaScript の構文解析が必要な点は，文献[5]と同様である．

6. 今後の課題

今後は，学習に用いるデータについて更に精査を行って学習を行いたい．検証に用いた学習データのうち悪性 JavaScript としたデータは，MWS Dataset2013 から D3M 攻撃通信データから一定の条件のもと精査した JavaScript を用いたが，データ自体の良悪性の判定は行っておらず，またデータの多様性についても検証を行なっていない．

学習結果に大きな影響を与える学習データ自体の検証は今後の課題となる. さらに多くの難読化した悪性 JavaScript を収集し，教師データとすることで本稿の検証とは違った結果が得られることも考えられる．

本稿では，難読化した悪質 JavaScript の検知を目的として SVM を用いて判定を行うことを提案した．しかし，本稿の検証で得られた SVM による学習の結果が，難読化の特徴を獲得したものかについては検証が出来ていない．本手法によって，難読化の有無に関係なく悪質な JavaScript の特徴点を学習することができる可能性がある．

そこで，本提案手法を用いて，難読化されていない悪質

な JavaScript も教師データに加えて学習することや，同じ難読化手法で難読化した一般の JavaScript と悪質な JavaScript を識別してみるなどの検証を行うことによって，文字出現頻度を SVM のパラメータとすることで何が識別できているのかについて，詳細に調査を行う必要がある．

また，本稿では提案手法の特性上，学習データの最小サイズを 1KB と規定したが，1KB というサイズ自体には根拠はない. 小さいサイズの JavaScript は多く存在するため，どの程度のサイズまで本手法で扱えるのかは今後検証が必要である．ただし，ある一定サイズ以下の JavaScript については本手法では扱えないことは明白であるため，他の手法との組み合わせによる検知を検討していく必要がある．

謝辞 本研究は，平成 24 年度に情報通信研究機構から委託を受け実施した「ドライブ・バイ・ダウンロード攻撃対策フレームワークの研究開発」の成果の一部である．
ご協力頂いた皆様に，謹んで感謝の意を表する．

参考文献

[1] 神薗雅紀，畑田充弘，寺田真敏，秋山満昭，笠間貴弘，村上純一：マルウェア対策のための研究用データセット～MWS Datasets 2013～，CSS2013(MWS2013) (2013.10)
[2] 神薗雅紀，西田雅太，星澤裕二：動的解析を利用した難読化 JavaScript コード解析システムの実装と評価，MWS2010 (2010)
[3] Alexa Top 500 Global Sites, Alexa (online), available from <http://www.alexa.com/topsites> (accessed 2013-11-18)
[4] Chang, C. and Lin, C.: LIBSVM -- A Library for Support Vector Machines (online), available from <http://www.csie.ntu.edu.tw/~cjlin/libsvm/> (accessed 2014-01-17)
[5] 神薗 雅紀，西田雅太，小島恵美，星澤裕二：抽象構文解析木による不正な JavaScript の特徴点抽出手法の提案, 情報処理学会論文誌 Vol.54 No.1 349‐356 (2013)
[6] Likarish, P., Jung, E. and Jo, I.: Obfuscated Malicious Javascript Detection using Classification Techniques (online), available from <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.155.5680&rep=rep1&type=pdf> (accessed 2014-01-17)