抽象構文木に基づくネスト構造に関する特徴を用いた悪性JavaScript検知手法

佐野 涼太[1]　花田 真樹[2]　早稲田 篤志[2]　村上 洋一[2]　布広 永示[2]　折田 彰[3]　関口 竜也[3]

概要：ドライブバイダウンロード攻撃による被害は年々深刻化している．その攻撃の被害に遭う主な原因としては，Web サイトに埋め込まれた不正スクリプトによって，訪問者が悪性 Web サイトにアクセスさせられてしまうということが挙げられる．その不正スクリプトの多くは攻撃の検知を回避するための様々な手法が施されているため，このような悪性 Web サイトによる攻撃を高精度で検知する手法が求められている．本稿では，JavaScript を用いた不正スクリプトを高精度に検知するために，スクリプトの抽象構文木における出現キーワード（出現文字列）とその属性（変数や関数など），プログラムの構造的特徴（ネストの深さとその位置的な分布）を利用して，機械学習アルゴリズム（SVM, NBC, Random Forest）に基づく検知手法を提案し，評価する．

キーワード：ドライブバイダウンロード攻撃, 機械学習, JavaScript

A Machine Learning-based Method for Detecting Malicious JavaScript using Nested Structure based on Abstract Syntax Tree

Ryota Sano[1]　Hanada Masaki[2]　Waseda Atsushi[2]　Murakami Yoichi[2]　Nunohiro Eiji[2]
Orita Akira[3]　Sekiguti Tatsuya[3]

Abstract: Damage caused by drive-by download has been getting worse year by year. One of the main causes of the attack is that malicious scripts embedded in web sites force visitors to access malicious web sites. Also, many of the scripts use obfuscation to prevent detection. Therefore, it is required to detect such attacks with high accuracy. In this study, in order to detect malicious JavaScript code with high accuracy, we propose a machine learning method using not only statistical features, i.e. the number of keywords in the code and their attributes, such as variables and functions, but also structural features (nest depth) based on abstract syntax tree. In addition, the accuracies of the methods with different combinations of the features and machine learning algorithms, SVM, Native Bayes and Random Forest, are evaluated.

Keywords: Drive-by Download Attack, JavaScript, Machine Learning

1. はじめに

ドライブバイダウンロード攻撃による被害は年々深刻化している．この攻撃は現在流行しているランサムウェアの代表的な感染経路となっている．ドライブバイダウンロード攻撃の例を図 1 に示す．ドライブバイダウンロード攻撃では，まず，ユーザを改ざんされた Web サイト（以降，改ざんサイトと呼ぶ）にアクセスさせることにより，そのユーザをいくつかの中継サイトに誘導する．中継サイトにアクセスさせる意図として第三者の調査，解析を困難にさせることが挙げられる．最終的にユーザを脆弱性を突く攻撃コードが埋め込まれた Web サイト（以降，攻撃サイトと

[1] 東京情報大学大学院 総合情報学研究科
Graduate School of Informatics, Tokyo University of Information Sciences
[2] 東京情報大学 総合情報学部
Department of Information Sciences, Tokyo University of Information Sceinces
[3] 株式会社日立システムズ サイバーセキュリティリサーチセンター
Hitachi Systems, Ltd. Cyber Security Research Center

呼ぶ）に誘導させることで，ユーザはマルウェアをダウンロードさせられる．ドライブバイダウンロード攻撃の被害を受ける主な原因として，改ざんサイトに含まれる不正スクリプトによるリダイレクトや，容易に攻撃サイトを作成することができる Explot Kit などのツールによるものが多い．不正スクリプトの多くには JavaScript が利用され，主に中継サイト，攻撃サイトへのリダイレクト，Web やブラウザなどのプラグインの情報確認などに用いられる．

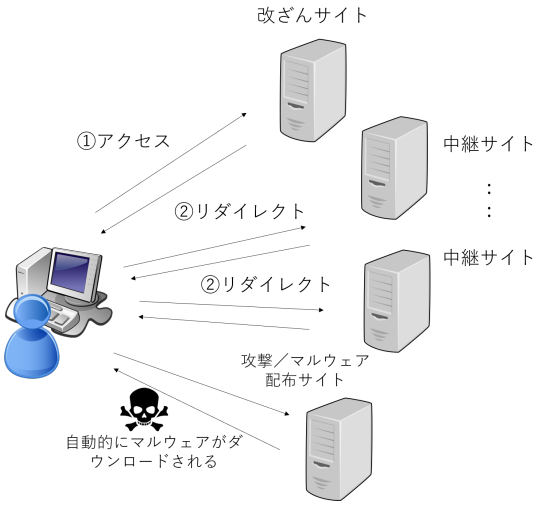

図 1　ドライブバイダウンロード攻撃の例
Fig. 1　Examples of drive-by download attacks

本研究では，次に挙げる 4 つの特徴を取り入れた悪性 JavaScript 検知手法を提案する．1 つ目として，悪性 JavaScript に出現するキーワードと悪意のない JavaScript（以降，良性 JavaScript と呼ぶ）には類似性に差異が現れると仮定し，出現キーワードを特徴とする．2 つ目として，良性 JavaScript と悪性 JavaScript には同じキーワードでも異なる意味として用いられていると仮定し，（出現キーワードに対する）属性を特徴とする．3 つ目として，良性 JavaScript と悪性 JavaScript ではプログラム構造（ネストの深さ）に特徴が現れるとして，ネストの深さを特徴とする．4 つ目として，3 つ目のプログラム構造（ネストの深さ）の特徴を拡張し，ネストの深さのプログラムにおける位置的な傾向を特徴とする．

本稿では、**2** で本研究に関する関連研究について述べる．**3** では，提案手法について述べる．**4** では．評価実験での結果と提案手法の有効性を評価する．**5** では，まとめと今後の課題について述べる．

2.　関連研究

本章では，本研究の関連研究について説明する．これまでに，難読化された JavaScript の類似性に着目し，悪性 JavaScript を検知・分類する手法が提案されている．

[1] は，機械的に生成された悪意ある JavaScript として，使われる文字列や値などは異なるが，構造自体には変化しない点に着目している．例として，悪性 JavaScript に出現する unscape 関数や replace 関数などの引数の値や変数名の変更によって難読化のパターンが変更されているが，JavaScript の構造自体は類似しているため，これらの特徴を用いて悪性 JavaScript の分類を行っている．[2] では，[1] を発展させた研究として JavaScript 構造だけでなく，それに含まれる属性も考慮せずに JavaScript の形状のみに着目して検知・分類を行っている．[3] では，JavaScript から長さ n の部分文字列の n-gram を生成し，その出現頻度を特徴とし，機械学習アルゴリズムを用いて検知を行っている．[4] では，JavaScript 内の文字出現頻度において悪質な JavaScript には難読化アルゴリズム自体に特徴が現れるとしている．難読化された JavaScript には文字列の置換処理などにより，難読化を解除したのち実行を行うため，難読化が施されてないものと比較して，文字出現頻度に差異が現れる傾向がある．これらの文字出現頻度の特徴を利用し，悪性 JavaScript の検知を行っている．[5] では，自然言語処理などで用いられる Paragraph Vector を用いて悪性 JavaScript の類似度を算出し，悪性 JavaScript の分類・可視化を行っている．[6] では，悪性 JavaScript と良性 JavaScript に出現するキーワード，変数名や関数名などの属性，ネストの深さの 3 つの特徴を用いて悪性 JavaScript の検知を行っている．本研究は，[6] を拡張したものであり，[6] で用いられている 3 つの特徴の詳細については，**3** で述べる．

[1], [2], [3] では，構文木を用いている点では似ているが，難読化された JavaScript のみに着目し分類を行っている．[4] では，難読化された JavaScript のみを用いて，文字の出現頻度に着目し悪性 JavaScript の検知を行っている．[5] に関しては，Paragraph Vector を用いた類似度に着目し分類・可視化を行っている．

3.　提案手法

本章では，JavaScript の抽象構文木における出現キーワードとその属性，ネストの深さ，ネストの深さのプログラムにおける位置的な傾向に関する 4 つの特徴を利用した，悪性 JavaScript の検知手法について述べる．まず，既に我々の提案している 3 つの特徴を述べたのち，本稿で新たに提案するネストの深さのプログラムにおける位置的な傾向に関する特徴を述べ，特徴ベクトルと機械学習アルゴリズムについて述べる．

3.1　抽象構文木における出現キーワードとその属性，ネストの深さに関する特徴情報 [6]

既に我々の提案している 3 つの特徴である，抽象構文木における出現キーワードとその属性，ネストの深さ [6] について簡単に述べる．

```
var a = 6 * 7;
```

図 2　JavaScript コード
Fig. 2　JavaScript code


```
{
  "type": "Program",
  "body": [
    {
      "type": "VariableDeclaration",
      "declarations": [
        {
          "type": "VariableDeclarator",
          "id": {
            "type": "Identifier",
            "name": "a"
          },
          "init": {
            "type": "BinaryExpression",
            "left": {
              "type": "Literal",
              "value": 6,
              "raw": "6"
            },
            "operator": "*",
            "right": {
              "type": "Literal",
              "value": 7,
              "raw": "7"
            }
          }
        }
      ],
      "kind": "var"
    }
  ],
  "sourceType": "script"
}
```


図 3　JavaScript の抽象構文木
Fig. 3　Abstract syntax tree of the JavaScript code

AST ツール (Babylon)[8] を使用して，JavaScript の抽象構文木（JSON 形式）を作成する．この JavaScript の抽象構文木（JSON 形式）から，出現キーワード，属性，入れ子の深さ（階層数）を抽出する．図 2 の JavaScript コードに対し，AST ツールを使って出力した例を図 3 に示す．JSON 形式では，図中の":"（コロン）の左側をキー，右側を値と呼ぶ．

3.1.1　出現キーワード

出現キーワードとして，JavaScript の抽象構文木（Jason 形式）の"name"，"operator"，"raw"キーの値を抽出する．図 3 の例では，"a"，"*"，"6"，"7"の 4 つがキーワードとして抽出される．加えて，JavaScript の抽象構文木（JSON 形式）において，表 1 の左側の「"type"の値」に示される文字列が表れた場合，表 1 の右側の文字列をキーワードとして抽出する．これにより，図 3 の例からは，"a"，"*"，"6"，"7"，"var"の 5 つがキーワードとして抽出される．JavaScript コード検体毎に，これらのキーワードをカウントし，出現回数を求める．

表 1　"type"の値に対するキーワード
Table 1　Keyword for the value of "type"

"type"の値	キーワード
VariableDeclaration	var
FunctionDeclaration	function
NewExpression	new
ReturnStatement	return
BreakStatement	break
ContinueStatement	continue
IfStatement	if
ForStatement, ForInStatement, ForOfStatement	for
WhileStatement	While
SwitchStatement	Switch
ThrowStatement	throw
TryStatement	try
CatchClause	catch

表 2　"type"の値（"operator", "params", "arguments"）に対する属性番号
Table 2　Attribute number for the value of "type", and "operator", "params" and "arguments"

"type"の値（"operator", "params", "arguments"）	属性番号
表 1 の"type"の値	1
VariableDeclarator, SwitchStatement ReturnStatement，IfStatement AssignmentExpression, ConditionalExpression LogicalExpression, BinaryExpression UnaryExpression, UpdateExpression ObjectExpression, ArrayExpression ThisExpression, ObjectPattern AssignmentPattern, ArrayPattern	2
FunctionExpression, CallExpression FunctionDeclaration	3
NewExpression	4
Literal, operator	5
params, arguments	6
ThrowStatement, CatchClause	7
その他	8

3.1.2　出現キーワードとその属性

抽出したキーワード（**3.1.1**）に対して，その階層に定義された"type"の値を属性を抽出する．なお，本研究では，表 2 に示すように，"type"の値に対する識別値（以降，属性番号とよぶ）を割り当て，属性を 8 種類に分類する．これにより，図 3 の"a"は属性番号 2，"*"は属性番号 5，"6"は属性番号 5，"7"は属性番号 5，"var"は属性番号 1 となる．JavaScript コード検体毎に，これらのキーワードとその属性の組み合わせた出現数をカウントし，出現回数を求める．

3.1.3　入れ子の深さ（階層数）

抽出した出現キーワード（とその属性）に対して，入れ子の深さ（階層数）を抽出する．なお，表 3 に示すキー"type"の値（条件分岐と繰り返しなど）のみに対し，階層数をカウントする．JavaScript コード検体毎に，キーワードとそ

表 3　階層数をカウントする”type”の値
Table 3　The value of ”type” for couting the number of layers

”type”の値
WhileStatement
DoWhileStatement
ForStatement
ForInStatement
ForOfStatement
IfStatement
SwitchStatement
FunctionDeclaration
ThrowStatement
TryStatement

の属性に対する階層をカウントし，階層数を求める．本研究では，JavaScript コード検体の入れ子の深さ（階層数）に関する特徴として，JavaScript コードで出現した階層数の中央値を用いる．これにより，図 3 の JavaScript コード検体では，表 3 に示すキー”type”の値（条件分岐と繰り返しなど）が存在しないため，入れ子の深さ（階層数）は 0 となる．

3.2　抽象構文木におけるネストの深さのプログラムにおける位置的な傾向に関する特徴情報

本研究では，新たな特徴として，ネストの深さのプログラムにおける位置的な傾向に関する特徴を用いる．ネストの深さのプログラムにおける位置的な傾向とは，抽象構文木を探索した場合の出現キーワード（とその属性）に対する入れ子の深さ（階層数）の分布の傾向を意味する．

下記に，出現キーワード（とその属性）に対する入れ子の深さ（階層数）の分布に関する特徴抽出手法を示す．

(1) 入れ子の深さ（階層数）を保持する変数 *depth* を 0 で初期化する．
(2) 抽象構文木を探索し，表 3 に示すキー”type”の値（条件分岐と繰り返しのステートメントなど）が出現したら変数 *depth* をインクリメントし，これらのステートメントが終了したら，変数 *depth* をデクリメントする．加えて，探索の過程で，キーワード（**3.1.1**）が出現したら，変数 *depth* を配列 *data_series* に順番に格納する．

図 4 の JavaScript コードに対し，AST ツールを使って出力した例を図 5 に示す．この抽象構文木（JSON 形式）に対して，上述の特徴抽出手法より得られるネストの深さの分布を示す配列 *data_series* は，$\{0, 0, 0, 0, 0, 0, 0, 1, 1, 1\}$ となる．配列 *data_series* の各要素の意味としては，キーワード”var”の入れ子の深さは 0，次のキーワード”a”の入れ子の深さは 0，次のキーワード”6”の入れ子の深さは 0，次のキーワード”*”の入れ子の深さは 0，次のキーワード”7”の入れ子の深さは 0，次のキーワード”while”の入れ子の深さは 0，次のキーワード”true”の入れ子の深さは 0，次のキーワード”a”の入れ子の深さは 1，次のキーワード”+=”の入れ子の深さは 1，次のキーワード”1”の入れ子の深さは 1 となる．

本研究では，上述の特徴抽出手法より得られるネストの深さの分布である配列 *data_series* を先頭から順番に 10 分割し，分割した各グループの最大値を特徴とする手法と，その最大値より求められる歪度を特徴とする手法の 2 つを用いる．

3.2.1　ネストの深さの分布（分割）

ネストの深さの分布である配列 *data_series* を先頭から順番に 10 分割し，分割した各グループの最大値を特徴（代表値）とする．図 5 の例では，データ数が 10 個なので，10 分割すると各グループのデータ数が 1 個なので，$\{0, 0, 0, 0, 0, 0, 0, 1, 1, 1\}$ が特徴ベクトルとなる．

3.2.2　ネストの深さの分布（歪度）

ネストの深さの分布である配列 *data_series* を先頭から順番に 10 分割し，分割した各グループの最大値を代表値とする．この代表値から歪度を算出し，これを特徴とする．歪度とは，分布の状態が正規分布からどの程度，歪みがあるかどうかを表す指標であり，左に歪みがある場合には「正の値」を示し，右に歪みある場合には「負の値」を示す．本研究では，歪度が「負の値」である場合はネストの深い箇所がプログラムの後半部分に現れる傾向を，「正の値」である場合はネストの深い箇所がプログラムの前半部分に現れる傾向を示す．

歪度の計算式を下記に示す．

$$\text{歪度} = \frac{n}{(n-1)(n-2)} \sum_{k=1}^{n} \left(\frac{x_i - \overline{x}}{s} \right)^3$$

なお，n はサンプルサイズ，s は標準偏差，$\overline{x}$ は各データ x_i の平均値である．

図 5 の例では，歪度の値は 1.035 となる．

4.　評価実験

本章では，悪性 JavaScript の検知精度の観点から提案手法の有用性を示す．実験に使用する検体について述べ，実験で用いる評価指標，実験結果について，下記に記す．

4.1　使用検体

評価実験で使用する検体については，マルウェア人材対策ワークショップから提供されている MWS Dataset 2018 に含まれる Download Data by Marionette Dataset（D3M Dataset）を使用し，悪性 JavaScript 検体として 300 検体抽出した．良性 JavaScript については URL に go ドメインが付いている政府系サイトから収集を行い上位 300 検体を良性 JavaScript とする．収集方法については Google 検索エンジンでの検索オプション「:（検索演算子）」を用いて，クローリングを行い収集を行う．

```
var a = 6 * 7;
while (true) {
        a += 1;
}
```

図 4　JavaScript コード
Fig. 4　JavaScript code


```
{
  "type": "Program",
  "body": [
    {
      "type": "VariableDeclaration",
      "declarations": [
        {
          "type": "VariableDeclarator",
          "id": {
            "type": "Identifier",
            "name": "a"
          },
          "init": {
            "type": "BinaryExpression",
            "left": {
              "type": "Literal",
              "value": 6,
              "raw": "6"
            },
            "operator": "*",
            "right": {
              "type": "Literal",
              "value": 7,
              "raw": "7"
            }
          }
        }
      ],
      "kind": "var"
    },
    {
      "type": "WhileStatement",
      "test": {
        "type": "Literal",
        "value": true,
        "raw": "true"
      },
      "body": {
        "type": "BlockStatement",
        "body": [
          {
            "type": "ExpressionStatement",
            "expression": {
              "type": "AssignmentExpression",
              "operator": "+=",
              "left": {
                "type": "Identifier",
                "name": "a"
              },
              "right": {
                "type": "Literal",
                "value": 1,
                "raw": "1"
              }
            }
          }
        ]
      }
    }
  ],
  "sourceType": "module"
}
```


図 5　JavaScript の抽象構文木
Fig. 5　Abstract syntax tree of the JavaScript code

4.2　評価指標

機械学習については機械学習ソフトウェアである「Weka」を用いて評価を行っていく．機械学習アルゴリズムについては上述で示したナイーブベイズ（Naive Bayes），ランダムフォレスト（Random Forests），SVM（Support Vector Machine）の 3 つを用いる．評価指標に関しては TPR，TNR，FPR，FNR，F-Measure，MCC，AUC の 7 種類を用いる．TPR，TNR，FRR，FNR，F-Measure，MCC の計算式を以下に記す．F-Measure に関しては，MCC に関しては，機械学習に用いる使用データにおいて通常不均衡なデータである場合 AUC については，値が 1 に近いほど判別性能が高く，値が 0.5 の時ランダムとなる指標である．個々のパラメータについては　，また，交差検証については良性検体，悪性検体ともに 10 分割交差検証を行う．

$$(1)\quad \mathrm{TPR} = \frac{\text{悪性 JavaScript 検体を悪性として分類した数}}{\text{悪性 JavaScript 検体の総数}}$$

$$(2)\quad \mathrm{TNR} = \frac{\text{良性 JavaScript 検体を良性として分類した数}}{\text{良性 JavaScript 検体の総数}}$$

$$(3)\quad \mathrm{FPR} = \frac{\text{悪性 JavaScript 検体を良性として分類した数}}{\text{悪性 JavaScript 検体の総数}}$$

$$(4)\quad \mathrm{FNR} = \frac{\text{良性 JavaScript 検体を悪性として分類した数}}{\text{良性 JavaScript 検体の総数}}$$

$$(5)\quad \text{F - Measure} = \frac{2 \cdot \mathrm{Precision} \cdot \mathit{Recall}}{\mathrm{Precision} + \mathrm{Recall}}$$

$$(6)\quad \mathrm{MCC} = \frac{\mathrm{TP} \times \mathrm{TN} - \mathrm{FP} \times \mathrm{FN}}{\sqrt{(TP+FP)(FN+TN)(TP+FN)(FP+TN)}}$$

4.3　実験結果

異なる特徴（キーワード，属性，階層，歪度，分割）を組み合わせて作成した各方法（表 4）の検知精度について評価実験を行った．また，異なる機械学習アルゴリズム（SVM、NBC、RF）を用いて評価実験を行った．表 5 は SVM，表 6 はナイーブベイズ，表 7 はランダムフォレストを用いた実験結果である．

結果として，いずれの機械学習アルゴリズムにおいても，属性の特徴を加えない方法に比べて，属性の特徴を加えた方法（M5, M9 11, M13 16）の方が検知精度が高く，F-Measure, MCC, AUC が 10% 20%程度高い値を示した（表 5, 6, 7 で背景色が灰色になっている方法）．すなわち，キーワードの属性を考慮したその出現回数を用いることにより，検知精度が大幅に向上することが確認された．

新規に提案したネストの深さに基づく歪度と分割の特徴を用いた方法は，SVM と NBC を用いた場合，M5（キーワード + 属性）や M9（キーワード + 属性 + 階層）の検知精度を超えることはできなかった．しかしながら，RF を用いた場合，分割のみの特徴を加えた M11（キーワード + 属性 + 分割）は，F-Measure と MCC に関して，それぞれ 2%と 3%向上することができた．歪度の特徴については，歪度のみを用いた RF に基づく方法（M2）の AUC は 0.73 であったが，この特徴を他の特徴に組み合わせた場合，M5 や M9 の性能を超えるほどの検知精度に寄与することはなかった．

表 4　Method1〜16 での特徴
Table 4　Features of Method 1-16

Method	type
M1	キーワード
M2	歪度
M3	分割
M4	歪度+分割
M5	キーワード+属性
M6	キーワード+階層
M7	キーワード+歪度
M8	キーワード+分割
M9	キーワード+属性+階層
M10	キーワード+属性+歪度
M11	キーワード+属性+分割
M12	キーワード+歪度+分割
M13	キーワード+属性+階層+歪度
M14	キーワード+属性+階層+分割
M15	キーワード+属性+歪度+分割
M16	キーワード+属性+階層+歪度+分割

表 5　SVM での検知結果
Table 5　Detection results using SVM

Method	TPR	TNR	FPR	FNR	F-Measure	MCC	AUC
M1	0.8	0.813	0.187	0.2	0.805	0.613	0.807
M2	0.41	0.723	0.277	0.59	0.486	0.14	0.567
M3	0.693	0.573	0.427	0.307	0.654	0.269	0.633
M4	0.703	0.577	0.423	0.297	0.661	0.282	0.64
M5	0.913	0.947	0.053	0.087	0.929	0.86	0.93
M6	0.803	0.817	0.183	0.197	0.809	0.62	0.81
M7	0.803	0.837	0.163	0.197	0.817	0.64	0.82
M8	0.77	0.807	0.193	0.23	0.784	0.577	0.788
M9	0.913	0.947	0.053	0.087	0.929	0.86	0.93
M10	0.917	0.943	0.057	0.083	0.929	0.86	0.93
M11	0.91	0.937	0.063	0.09	0.922	0.847	0.923
M12	0.767	0.797	0.203	0.233	0.778	0.564	0.782
M13	0.92	0.943	0.057	0.08	0.931	0.864	0.932
M14	0.91	0.937	0.063	0.09	0.922	0.847	0.923
M15	0.91	0.927	0.073	0.09	0.918	0.837	0.918
M16	0.91	0.927	0.073	0.09	0.918	0.837	0.918

本研究では，ネストの深さの構造に着目した歪度と分割の特徴を提案した．

歪度に関しては，検知精度の向上に寄与が見られなかった．その要因として，歪度の特徴には JavaScript コードのネスト構造を 1 つの特徴として表してしまうため，歪みが反対でも同値になることにより，先行研究の特徴と比べて検知精度の向上はみられなかったのではないかと考えられる．

分割に関しては，1 つの値として表す歪度の値と異なり，ネスト構造の形状を特徴として扱うことができるため，結果として，RF では先行研究の特徴から 3 %の向上が見られた．しかしながら，SVM，NBC では検知精度の向上はみられなかった．

表 6　NB での検知結果
Table 6　Detection results using NB

Method	TPR	TNR	FPR	FNR	F-Measure	MCC	AUC
M1	0.513	0.873	0.127	0.487	0.626	0.414	0.727
M2	0.43	0.693	0.307	0.57	0.495	0.128	0.557
M3	0.783	0.447	0.553	0.217	0.67	0.244	0.625
M4	0.777	0.447	0.553	0.223	0.667	0.237	0.636
M5	0.71	1	0	0.29	0.83	0.742	0.916
M6	0.513	0.873	0.127	0.487	0.626	0.414	0.727
M7	0.513	0.873	0.127	0.487	0.626	0.414	0.727
M8	0.513	0.873	0.127	0.487	0.626	0.414	0.728
M9	0.707	1	0	0.293	0.828	0.739	0.916
M10	0.71	1	0	0.29	0.83	0.742	0.915
M11	0.71	1	0	0.29	0.83	0.742	0.915
M12	0.513	0.873	0.127	0.487	0.626	0.414	0.728
M13	0.707	1	0	0.293	0.828	0.739	0.915
M14	0.71	1	0	0.29	0.83	0.742	0.915
M15	0.71	1	0	0.29	0.83	0.742	0.915
M16	0.71	1	0	0.29	0.83	0.742	0.915

表 7　RF での検知結果
Table 7　Detection results using RF

Method	TPR	TNR	FPR	FNR	F-Measure	MCC	AUC
M1	0.7	0.813	0.187	0.3	0.742	0.517	0.844
M2	0.62	0.76	0.24	0.38	0.667	0.384	0.726
M3	0.637	0.82	0.18	0.363	0.701	0.465	0.784
M4	0.647	0.81	0.19	0.353	0.704	0.463	0.796
M5	0.92	0.99	0.01	0.08	0.953	0.912	0.993
M6	0.72	0.79	0.21	0.28	0.746	0.511	0.832
M7	0.717	0.837	0.163	0.283	0.762	0.557	0.848
M8	0.727	0.82	0.18	0.273	0.762	0.549	0.848
M9	0.913	0.99	0.01	0.087	0.95	0.906	0.994
M10	0.917	0.99	0.01	0.083	0.952	0.909	0.992
M11	0.937	0.997	0.003	0.063	0.966	0.935	0.993
M12	0.727	0.827	0.173	0.273	0.765	0.556	0.848
M13	0.923	0.99	0.01	0.077	0.955	0.915	0.992
M14	0.933	0.997	0.003	0.067	0.964	0.932	0.994
M15	0.933	0.993	0.007	0.067	0.962	0.928	0.994
M16	0.94	0.99	0.01	0.06	0.964	0.931	0.994

実験結果から出現キーワードのみを用いた時よりも，属性を特徴として取り入れることにより検知精度は大きく向上しているため，今後は更なる検知精度の向上を目的として，属性カテゴリーの再検討を行う．また，今回用いた分割では 10 分割を使用したが，何分割が最も良い結果となるのか詳細に調査が必要である．

5. まとめと今後の予定

本研究では JavaScript の抽象構文木において，出現キーワード，属性，入れ子の深さ（階層数）に加え，歪度，分割に関する特徴情報を利用して悪性 JavaScript 検知手法を提案した．政府系サイトと MWS D3M Dataset2018 を使用した評価実験を行い，悪性 JavaScript の検知精度の観点から提案手法の評価を行った．評価実験においては，3 種類

の機械学習アルゴリズムの中でランダムフォレスト（RF）の検知精度が他のアルゴリズムと比較して最も高い結果となり，高精度で検知が可能であることを示した．

今後の課題として，本研究では出現キーワード，属性，入れ子の深さ（階層数）を特徴情報として使用しているが，本評価実験としてランダムフォレスト（RF）が適している理由や使用する特徴間の依存関係に関する分析を進めると同時に，更なる検知精度向上のための新たな特徴を検討する予定である．また，評価実験に関しては主に次の課題が考えられる．

- 良性 JavaScript には Alexa[8] が公開しているサイトを使用する
- 使用検体数が少ないため，検体数を増やす．
- 使用していない機械学習アルゴリズムを適用する．

参考文献

[1] 神薗雅紀，西田雅太，小島恵美，星澤祐二，“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 コンピュータセキュリティシンポジウム 2012, pp.232-237, Oct..2012.

[2] 上西拓真，神薗雅紀，西田雅太，森井昌克，”抽象構文解析木の符号化による不正な JavaScript の分類手法の提案” 情報処理学会 コンピュータセキュリティシンポジウム 2012, pp.232-237, Oct. 2012.

[3] Aurore Fass, Robert P.Krawczyk, Michael Backes, Ben Stock, ”JaSt: Fully Syntactic Detection of Malicious (Obfuscated) JavaScript” Detection of Intrusions and Malware, and Vulnerability Assessment, pp.303-325, June, 2018.

[4] 西田雅太，星澤裕二，笠間貴弘，衛藤将史，井上大介，中尾康二，“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”, 情報処理学会 研究報告 マルチメディア通信と分散処理, pp.1-7, Feb.2014.

[5] 三須剛史，巻島和雄，岡田晃市郎，岩本一樹，“ソースコードの類似度に基づく悪性 JavaScript の分類に関する一検討” 情報処理学会 コンピュータセキュリティシンポジウム 2017, pp.378-384.

[6] 佐野涼太，佐藤順子，村上洋一，花田真樹，布広永示，”抽象構文木に基づく情報を用いた悪性 JavaScript 検知手法”，電子情報通信学会技術研究報告 信学技報, pp.63-68, Nov.2018.

[7] マルウェア対策研究人材育成ワークショップ 2018, https://www.iwsec.org/mws/2018/about.htm, 参照 Apr. 2018.

[8] Esprima, https://esprima.org/, 参照 Apr.2018.

[9] Babylon, https://github.com/babel/babylon, 参照 May.2018.

[10] Kali Linux, https://www.kali.org/, 参照 May.2018.

[11] Alexa Top 500 Global Sites, https://www.alexa.com, 参照 May.2019.