jjencodeで難読化されたJavaScriptの検知

平成 29 年度

学士学位論文

jjencode で難読化された JavaScript の検知

1180357　　中村 弘亮

指導教員　　清水明宏

2018 年 2 月 28 日

高知工科大学 情報学群

要 旨

jjencode で難読化された JavaScript の検知

中村 弘亮

近年，改ざんされた正規のウェブサイトなどを閲覧してしまうことより知らないうちに悪意のある Web サイトへと接続され，マルウェアに感染させられるというドライブバイダウンロード攻撃が報告されている．攻撃に使われるような悪意のある JavaScript のほとんどには難読化という処理が施されているが，一般的な Web サイトでも JavaScript に難読化を施すことがある．このように攻撃に用いられることもあるが悪意のない通常の Web サイトでも使用されうる難読化の研究は重要であるといえる．

様々な難読化手法の中でも比較的新しい手法に，記号のみのコードへ変換するというものがある．しかし，難読化後のコードが記号のみになる難読化手法を対象とした研究は未だ少ないという現状がある．この難読化手法には既存の研究である西田らの手法が有効であると考えられるが，西田らの手法には機械学習により獲得したものが何の特徴であるかの検証が十分ではないという課題があった．

そこで本論文では，難読化後のコードが記号のみの場合の難読化は検出可能かを検証することを目的とし，既存研究の西田らの手法である文字出現頻度を用いると何が判別可能であるのかの検証を行う．この実験により jjencode により難読化したコードと一般的なコードは判別が可能であることが示され，既存の手法では難読化が学習できることが明らかになった．

キーワード　JavaScript，難読化，jjencode

Abstract

Koryo NAKAMURA

In recent years, a drive-by download attack of infecting malware by browsing a tampered legitimate website has been reported. Most of the malicious JavaScript used for attacks has been treated as obfuscated, but obfuscation may be applied to JavaScript on common Web sites as well. Obfuscation research that can be used for attacks in this way but can also be used on ordinary malicious Web sites is important.

Among various obfuscation methods, there is a relatively new method of converting to a code with only symbols. However, there are still few studies targeting obfuscation method where obfuscated code becomes only symbols. Nishida's method, which is an existing research, is effective for this obfuscation method, but the method had a problem that verification of the feature acquired by machine learning was not sufficient.

In this paper, we aimed to verify whether obfuscation can be detected if the code after obfuscation is only a symbol. In addition, we verify what can be distinguished by using the character appearance frequency. This experiment showed that it was possible to discriminate between code obfuscated by jjencode and general code, and it became clear that obfuscation can be learned by using the existing method.

key words JavaScript, obfuscation, jjencode

目次

図目次

表目次

第 1 章

序論

1.1 背景と目的

近年，改ざんされた正規のウェブサイトや細工された不正広告を閲覧してしまうことより知らないうちに悪意のある Web サイトへと接続させマルウェアに感染させられるという被害が報告されている [1]．このような攻撃をドライブバイダウンロード攻撃といい，この攻撃には JavaScript が用いられている．

攻撃に使われるような悪意のある JavaScript のほとんどには難読化という処理が施されており，検知されることを回避しているが，私たちが日常的に訪れる一般的な Web サイトであっても難読化はコードのコピー防止などの目的により使用される可能性がある．そのため一概に難読化されているということが悪意のあるコードであるということに結び付くとは限らない．このように攻撃に用いられることもあるが悪意のない通常の Web サイトでも使用されうる難読化の研究は重要であるといえる．

難読化手法には様々なものがあるが，他の手法に比べ近年登場した手法に jjencode を用いるものがある [2]．これは演算により文字や数字を生成することでコードに記号以外を登場させないよう難読化できるというものである．比較的新しい手法であり，攻撃に用いられたことがあるにも関わらず，難読化後のコードが記号のみになる難読化手法を対象とした研究は未だ少ないという現状がある．このような難読化手法に対して有効であると考えられるものに，西田らの手法である文字出現頻度を利用した難読化 JavaScript の検出が挙げられる [3]．しかし，この検出手法は機械学習により獲得したものが難読化の特徴であるか，悪質なコードの特徴であるかの検証が十分ではないという問題点を抱えている．

そこで本研究では，難読化後のコードが記号のみの場合の難読化は検出可能かを検証することを目的とした．また，これにより既存研究の手法である文字出現頻度を用いると何が判別可能であるのかの検証をすることができるため，既存手法の課題となっている何が識別できているかについて明らかにしていく．

1.2　本論文の構成

本論文は全 6 章で構成される．本章では，難読化を研究すべき背景と本研究における二つの目的について述べた．第 2 章では，難読化を行う目的とその手法，さらに難読化の解析手法について述べる．第 3 章では，悪質な難読化 JavaScript を文字出現頻度を利用して検出する西田らの既存研究について述べ，そこにおける課題についても触れる．第 4 章では，記号のみが出現する難読化についての研究と既存研究の課題の解決を兼ねた提案より実験を行い，その結果を示す．第 5 章では，実験により得られた結果から考察を行う．そして第 6 章では，本研究の結論を述べ，さらに今後の展望を示す．

第 2 章

難読化

本章では，難読化の目的と難読化手法，さらにそれらのうちの悪意のある難読化された JavaScript を検出する解析手法について説明する．

2.1 難読化の目的

難読化とは，コードをできる限り読みづらくなるようコードの表現を改めることであり，主に次に説明する 4 つの目的を達成するために施される．

表 2.1 には，悪意のあるコードか一般的なコードかで使われるかを目的別にまとめている．表からもわかるように，意図の隠蔽を目的とした場合は，悪意のあるコードと一般的なコードでの使用が予想される．ただし，これらの目的のうち一つだけを理由として難読化されることもあれば，複数個の目的のために難読化されることもある．

表 2.1 使用目的の種類

目的	悪意のあるコードでの使用	一般的なコードでの使用
検知の回避	○	×
意図の隠蔽	○	○
コピー防止	×	○
変更の阻止	×	○

2.1.1　検知の回避

難読化はパターンマッチングなどの検知を回避するために使われることがある．難読化を行うことによりコードの表現が変わるため，単語での検知などは困難となる．ただし，普通この目的での使用は，悪意のあるコードを侵入させようとするときのものである．

2.1.2　意図の隠蔽

自分が考えた処理手順を真似されたくないときや処理を理解されることで処理結果を予想されることを避けたいときは，処理の意図を隠すために難読化されることがある．実行されようとしているコードが実際に何をしようとしているのかを知りたい場合，コードを読まなければならないため，難読化を施すことにより処理を理解することは困難となる．

2.1.3　コピー防止

難読化はコピーアンドペーストを防ぎたいという場合に使われることがあり，この目的では一般的な Web サイトに使用されうるといえる．

コピーアンドペーストしたコードを別の場所で動かしたい場合，通常であればそのコードに元の Web サイト特有のもの，例えば，ドメイン名や HTML タグの id・属性・内容などを取得する部分があると，ペーストした場所の環境にあったものに置き換えるか，そもそもその部分のコードを削除する必要がある．しかし，難読化されている場合，そういったことが非常に難しくなるため，ペーストした場所の環境では動かなくなることが期待できる．

2.1.4　変更の阻止

難読化を施すことにより，難読化後のコードのある部分が元のコードのどこに当たるのかを，解読しなければわからないようにできる．このことが，難読化後のコードの一部を変更すると元コードの予想もしないところに影響を与えてしまい，問題のない実行ができなくなる可能性に繋がるため，実行可能なコードを取得されたとしても，それを簡単に変更できな

いようにできる．

2.2　難読化手法

難読化には様々な手法が存在するため，ここではその一部を紹介する．さらに，本研究で用いた jjencode が用いる難読化の方法についても説明する．

難読化手法も難読化を行う目的と同じように，ただ一つの手法が使われることもあれば，複数の手法が織り交ぜられることがある．例として，図 2.1 に示すコードを JavaScript Obfuscator Tool[4] という様々な手法でコードを難読化することのできるツールを用いた場合，どのようなコードになるのかを図 2.2 に示した．図 2.1 は，デバッガのコンソール上に `Hello World!`と表示させるコードであり，図 2.2 のコードは複数の難読化手法により難読化が施されたコードである．図 2.1 のコードに難読化を施すと図 2.2 のようなコードになり，見ただけでは何を行うのかを理解することが困難となっていることがわかる．

```
console.log("Hello World!");
```

図 2.1　難読化前のコード

```
var _0x336d=['\x48\x65\x6c\x6c\x6f\x20\x57\x6f\x72\x6c\x64\x21'];(function(_0x3
3707a,_0x2179f4){var _0x3ff0cc=function(_0x5ce34e){while(--_0x5ce34e){_0x33707a
['push'](_0x33707a['shift']());}};_0x3ff0cc(++_0x2179f4);}(_0x336d,0x1d3));var
_0x1c49=function(_0x2e4bcb,_0x315381){_0x2e4bcb=_0x2e4bcb-0x0;var _0x3aac0c=_0x
336d[_0x2e4bcb];return _0x3aac0c;};console['\x6c\x6f\x67'](_0x1c49('0x0'));
```

図 2.2　難読化後のコード

また，複数の難読化手法を様々な部分に施すことの他，ある手法で難読化を行った後に，別の手法でさらに難読化を行うことができるため，特定の難読化手法を行った後のコードが検知されたとしても，その亜種などが登場しやすいといえる．

2.2.1　変数や関数名の変更

変数や関数の名前をランダムで長い名前や一文字にすることで動作をわかりづらくすることができる手法である．また，難読化前は目的別に用意されている変数を難読化後はいくつかの変数を使い回すよう変更することで，変数の用途が一つに定まらなくなり，より解読が困難となる．

2.2.2　実行されないコードの埋め込み

難読化後のコード内に実行されないコードを埋め込む手法である．難読化後のコードが直接理解できるようなものでないことが利用でき，コード全体の解読や把握を遅らせ，元のコードの推測を困難なものにすることができる．

2.2.3　文字列リテラルの変換

文字列のリテラルを分割することや Unicode など別の表現に置き換える手法である．これによりコード理解するためのヒントとなる可能性がある，コード中に出てくる文字列リテラルを一時的ではあるが読めなくすることができる．また，意味のある文字列リテラル中に意味のない文字をいくつも挿入することにより，一度読めなくし，コードが実行される段階で，文字列リテラル中の意味のない文字以外を抽出するいわゆる復号を行う関数が搭載されているという巧妙な変換を行うものも存在している．

2.2.4　演算による文字列や数字の生成で置き換え

難読化前に存在した文字列や数字を難読化後は演算により代替した等価なものに置き換える手法である．JavaScript は文字列と数字間を演算子により型変換することができる．次に示す図 2.3，2.4，2.5 のコードは，演算によって文字列と数字を生成できる JavaScript の例である．

図 2.3 は −1 という数字を生成できるが，コード中に数字が登場しない．図 2.4 は文字列

```
$=~[];
```

図 2.3　“-1” という数字を生成する演算

```
$=(![]+"");
```

図 2.4　“false” という文字列を生成する演算

の false を，図 2.5 は文字列の f を生成しているが，図 2.3 と同じように図 2.4，2.5 にも数字や文字が登場しないことがわかる．このような難読化と変数名などにも記号を用いるということを行うことにより，図 2.6 のような記号しか登場しないコードにすることができる．図 2.6 は jjencode と呼ばれるコードが全て記号のみとなる難読化を施したコードの一部である [5].

2.3　解析手法

難読化された悪意のある JavaScript を検出する手法としては，動的解析と静的解析があり [6]，それぞれに特徴がある．ここではその解析で対応できる主な JavaScript コードについても述べる．

2.3.1　動的解析

動的解析とは JavaScript を実際に動かし，その挙動を調べる方法である．難読化されているか否かに関わらず動かすことができれば解析の対象とすることができる．しかし，実行するため，実行環境によっては危険が伴うこともある．また，本来動かそうとしている環境

```
$=(![]+"")[0];
```

図 2.5　“f” という文字列を生成する演算

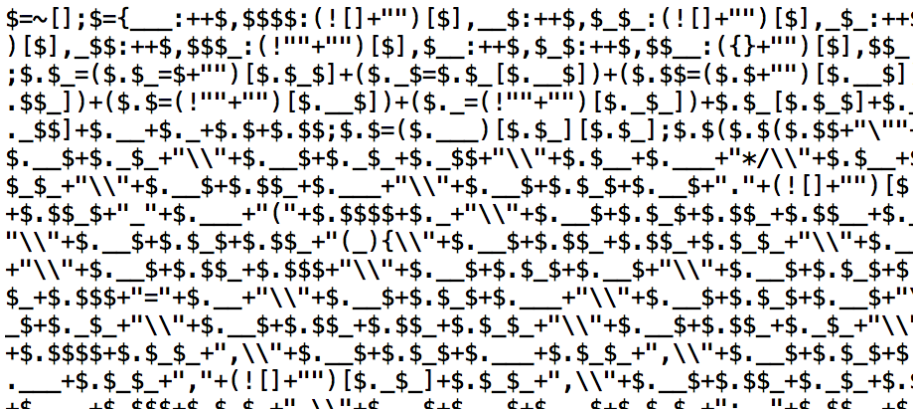

図 2.6　jjencode で難読化を施したコードの例

に限りなく近づけた環境を用意して安全に動かすといったことも可能だが，そうなると解析に少なからず手間がかかってしまう．

2.3.2　静的解析

静的解析とは JavaScript のソースコードを用いて，そのコードを調べる方法である．難読化されているか，されていないかによってソースコードは大きく変わるため，そこを利用する．scirpt タグ以外の要素内にコードの本体を置き，JavaScript ではそれを取得して実行するというようなコードに対しては結果を期待できないが，動かすことを必要としないため，コードが不完全であっても解析を行うことができる．本研究で扱うのはこちらの静的解析である．

第 3 章

既存研究

本章では，既存研究である西田ら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 の検出について説明し，本論文の目的に関わる既存研究の課題となっている点についても述べる．

3.1 悪意のある JavaScript の特徴

悪意のある JavaScript はそのほとんどに難読化が施されている．難読化が施されているため，そのコードの実際の挙動を予想することは難しいと言える．しかし，難読化後のコード自体の文字出現頻度はその手法による特徴として扱うことができる．

特徴が文字出現頻度に現れることを利用し，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 の検出が西田らによって提案されている．

3.2 西田らの手法

悪意のある難読化コードと一般的なコードを判別できる手法であり，特徴としては，文字出現頻度のみを特徴パラメータとするため計算コストが小さいこと，コードが不完全であっても検知対象とできることが挙げられる．

悪意のあるコードとして，MWS Dataset 2013[7] 内の D3M 攻撃通信データのうち，明らかに難読化されているコードを抽出している．また，一般的な Web サイトで使われるコードとして Alexa[8] が公開しているアクセス数トップ 500 の Web サイトより，それらのトップページで用いられている JavaScript を取得している．

ただし，既存研究ではファイルサイズが 1KB 未満のものは，そのコードの特徴がうまく現れないことを予想して除外している．

3.3　文字出現頻度

文字出現頻度は ASCII 文字の 0x21 から 0x7e までの 96 種類の文字をベクトルとして 96 次元で定義する．文字出現頻度の対象とした文字種類を表 3.1 に示した．

表 3.1　対象とした文字種類

文字の種類	対象範囲	16 進数表示
アルファベット	[A-Z]	0x41-0x5a
	[a-z]	0x61-0x7a
数字	[0-9]	0x30-0x39
記号	アルファベット，数字，空白，制御文字以外	0x21-0x2f，0x3a-0x40，0x5b-0x60，0x7b-0x7e

文字 i の出現数を m_i とするとき，対象文字数 N と文字出現頻度 $F(i)$ は，それぞれ次の (3.1)，(3.2) で定義される．

$$N = \sum_{i}^{n} m_i \tag{3.1}$$

$$F(i) = \frac{m_i}{N} \tag{3.2}$$

ただし，文字出現頻度 $F(i)$ は確率で表されるため 0 から 1 の範囲に収まる．このため，正規化は行わない．

3.4 SVM による学習

SVM(Support Vector Machine) は，マージンの最大化を実現するパターン認識の方法である．入力ベクトルに対してマージンを最大化するような超平面を求めることで 2 クラスの分類をすることができる [9].

96 種類の文字をベクトルとした 96 次元を SVM の入力ベクトルとして機械学習を行なった.

3.5 西田らの研究における課題

SVM による学習の結果，RBF(ガウス) カーネル，$C = 25.22$，$\gamma = 55.72$ のとき 98.84%という正答率 (accuracy) で分類ができたことから悪意のある難読化コードと一般的なコードを判別可能であったということができる．しかし，悪意のある難読化コードと一般的なコードの判別において，機械学習により獲得したものが難読化の特徴であるか，悪質なコードの特徴であるかの検証ができていないという課題が存在している.

第 4 章

課題解決の提案

前章で述べた西田らの研究における課題は，悪意のある難読化コードと一般的なコードの判別において，機械学習により獲得したものが難読化の特徴であるのか，悪質なコードの特徴であるのかの検証ができていないというものである．また，jjencode に代表される難読化した後のコードを記号のみのコードに変換することができる難読化に関する研究が少ないという現状がある．

そこで本研究では，既存研究の文字出現頻度を入力パラメータとした機械学習による判別を一般的なコードの難読化していないものと jjencode で難読化したものに適用した．これにより，難読化後のコードが記号のみの場合は難読化が検出可能であるかを検証することができ，さらに，既存研究の文字出現頻度を用いた機械学習では何が判別可能であるのかを明らかにすることができる．

4.1 実験

実験を以下の手順で行った．次にそれぞれの手順について詳しく説明する．

1. 一般的な Web サイトのコードの収集
2. 収集したコードの半数を jjencode により難読化
3. 文字出現頻度の算出
4. データのラベルの付与
5. SVM による学習と検証

4.1.1 一般的な Web サイトのコードの収集

既存研究で一般的な Web サイトのコードとして Alexa Top 500 Global Sites が利用されており，本研究でもその Web サイトからコードを取得した．

コードは Alexa の上位 10 サイトのトップページにアクセスし，script タグで囲まれた JavaScript と script タグにより読み込まれる JavaScript で記述されたファイルの内容を対象としている．これにより JavaScript を 438 個を収集した．表 4.1 に取得した一般的な Web サイトで用いられている JavaScript のデータについて示した．

表 4.1 一般的な JavaScirpt のデータ

個数	438 個
合計	14,754,808 bites
平均	約 33,687 bites
最大	1,682,739 bites
最小	23 bites

既存研究では，1KB 未満のファイルサイズであればその JavaScript を除外していた．これは，JavaScript の特徴が文字数が少ない場合に現れにくいことを危惧したための処理である．しかし，jjencode で難読化する場合，難読化後は記号のみになるという特徴が現れることが予想できる．このため，本実験ではファイルサイズが 1KB 未満である場合でも取得の対象としている．

注意点として，ここで取得した実際に一般的な Web サイトに用いられているコードは悪意のないコードであるとした．

4.1.2 収集したコードの難読化

収集したコードのうち半数を jjencode により難読化を施した．

既存研究では，悪意のある難読化されたコードとして MWS Dataset 2013 内の D3M 攻

撃通信データのうち，明らかに難読化されているコードを選んでいるが，その中に無害な難読化コードを含んでいる可能性を排除できていない．

本研究では，実際に一般的な Web サイトに用いられているコードから 2 種類のデータを準備するため，文字出現頻度を用いた機械学習により難読化自体が判別できるのかを検証することができる．

4.1.3　文字出現頻度の算出

文字出現頻度としては ASCII 文字のうち制御文字と空白文字を除く文字コード 0x21 から 0x7e までのアルファベット，数字，記号の計 94 種類の文字を対象とする．対象とした文字の総数からそれぞれの文字種別に文字数を割り，文字が出現する確率とした．また，既存研究と同じくここでは，算出した文字出現頻度は確率であり，0 から 1 の範囲に収まるため正規化は行わない．

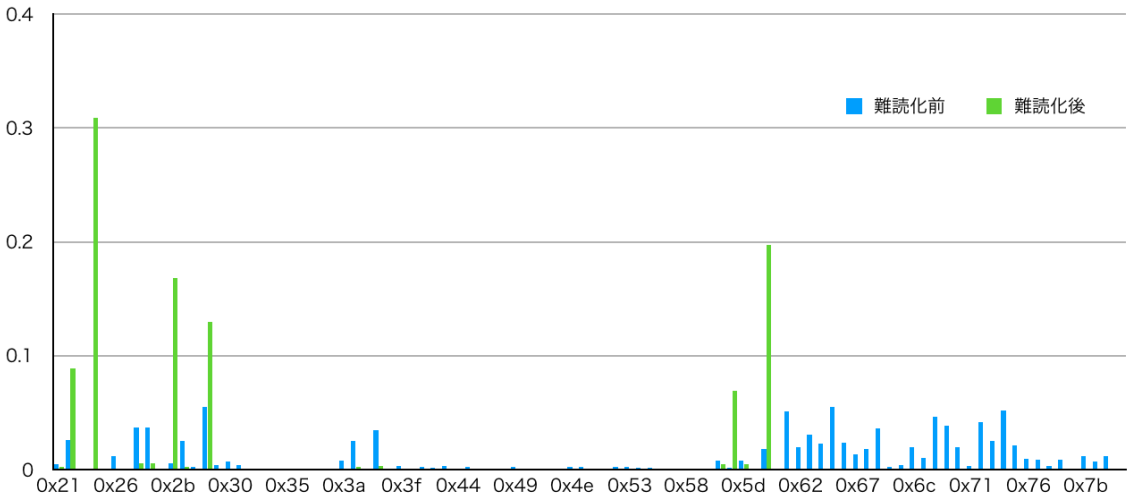

図 4.1　難読化前と後の文字出現頻度

図 4.1 はあるコードを jjencode により難読化する前と後の文字出現頻度を表したグラフである．横軸は，左から右に 0x21 から 0x7e までの文字種類を表しており，縦軸は算出した文字出現頻度が確率で表されている．

表 4.2　文字出現頻度上位 5 つの文字

難読化前	文字種類	. (0x2e)	e (0x65)	t (0x74)	a (0x61)	n (0x6e)
	出現確率	0.055343	0.055101	0.051603	0.051199	0.046320
難読化後	文字種類	$ (0x24)	_ (0x5f)	+ (0x2b)	. (0x2e)	” (0x22)
	出現確率	0.309287	0.196983	0.167870	0.129734	0.088579

表 4.2 は図 4.1 で示された文字出現頻度のうち，出現確率が上位 5 つの文字種類と確率を示したものである．

この図や表より，jjencode による難読化を用いると文字出現頻度に特徴が現れていることがわかる．

4.1.4　データのラベルの付与

機械学習の際に必要なデータにラベルを付与する．これまでの手順で準備した一般的な Web サイトのコードと jjencode により難読化を施したコードの合計 438 個のコード全てに，難読化を施したか施していないかのラベルを付与した．

4.1.5　SVM による学習と検証

準備した文字出現頻度のデータを用いて SVM により機械学習を行った．入力ベクトルは対象とした文字種類数に合わせて 96 次元となる．さらに，検証のため 10 分割交差検定を行った．

4.1.6 実験結果

今まで述べた手順に従い実験を行った結果，RBF カーネル，$C = 1$，$\gamma = 1$ のとき分類の正答率は 100%となった.

第 5 章

考察

実験結果より，難読化後に記号しか登場しない jjencode による難読化は，難読化を行っていない一般的なコードと判別できることがわかった．また，難読化コードと一般的なコードの判別ができたため西田らの手法は難読化を学習できたということができる．

今回の実験では，正答率が 100%という結果を得ることができた．jjencode ではコード中に記号以外を登場させないよう難読化を施すため，その特徴的な難読化方法により判別が正答率 100%で行えたと考えられる．また，既存研究で行われていた 1KB 未満のファイルサイズである場合除外するという操作は，難読化手法によっては行わなくても判別が可能であることが示された．

また，難読化後のコードが記号のみになる場合であっても難読化を検知することが可能であったことから jjencode の亜種にあたる，難読化後のコードが記号を使う顔文字となる aaencode などの難読化に対しても，同じような方法を用いて検出できる可能性がある．

今後の展望としては，本研究で行った静的解析による難読化そのものの検出の研究は悪意のある難読化を検出するという研究よりも少ないため，難読化の判別を研究として行っていく必要があると考える．

静的解析はその解析方法から悪意があるかないかより難読化されているのかいないのかを調べるのに向いており，動的解析はコードに悪意があるのか調べるために使うのが適している．このため，難読化されているか否かを静的解析で検出することができれば，検出されたコードのみを対象として動的解析を行うというシステムを構築することは難しくない．そのようなシステムを利用すれば，訪れた Web サイトにおいて静的解析した場合に難読化が施されている可能性があれば実行しないでおき，どうしても実行しなければならない場合のみ

動的解析に回して悪意のあるコードでないかを検査するということも可能である．こうすることで，動的解析で訪れる全ての Web サイトで行うなどといったことを避けることができる．

また，静的解析した場合に，よく利用する Web サイトで初めて難読化が検出され，それが不自然な場合は，Web サイトの管理者にそれを知らせることも可能である．さらに，Web サイトの管理者は管理している全ページを静的解析により一通り解析することにより，検知を逃れるよう細工されたコードが挿入されていないかを確認することもできる．

このような利点から，静的解析による難読化の研究は今後もより研究されなければならないと考えた．

第 6 章

結論

本研究では、難読化後のコードが記号のみの場合は難読化が検出可能であるかを検証し，既存研究の文字出現頻度を用いた機械学習では何が判別可能であるのかを明らかにした．

既存研究である西田らの文字出現頻度を入力パラメータとした機械学習による判別を jjencode により難読化したコードが一般的なコードと判別ができるかの実験に適用した．こうすることで，既存研究の課題であった検証とともに難読化して生成されるコードが記号のみとなる難読化手法に関する研究を行うことができた．

また，この実験により jjencode により難読化したコードと一般的なコードは判別が可能であることが示され，西田らの手法で難読化が学習できることが明らかになった．

さらに，難読化後のコードが記号のみになる場合であっても難読化を検知することが可能であったことから jjencode の亜種などにも本研究と同様の方法でを用いて検出できる可能性があるといえる．

今後の展望として，本研究で行った静的解析による難読化そのものの検出の研究は悪意のある難読化を検出するという研究よりも少ないという現状ではあるが，静的解析による難読化そのものの検出はユーザの負担を減らすことができる難読化を検知するシステムなどに応用可能である．そのため，静的解析による難読化の判別を研究として行っていく必要があると考えられる．

謝辞

本研究の遂行と論文作成にあたり，ご指導，ご助言いただきました高知工科大学情報学群 清水明宏教授に心より感謝し厚く御礼申し上げます．本研究の副査を担当していただきました高知工科大学情報学群 敷田幹文教授，吉田真一准教授に深く御礼申し上げます．最後に，有益な議論を交わしていただいた高知工科大学 清水研究室の皆様に深く感謝いたします．

参考文献

[1] Information-technology Promotion Agency, Japan（IPA）, “2016 年 1 月の呼びかけ: IPA 独立行政法人 情報処理推進機構,” https://www.ipa.go.jp/security/txt/2016/01outline.html, 2018.

[2] Yosuke HASEGAWA, “JavaScript 難読化読経,” https://www.slideshare.net/hasegawayosuke/javascript-51570525, 2018.

[3] 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, ” Vol.2014-CSEC-64, No.21, pp.1-7, 2014.

[4] Tiago Serafim, “JavaScript Obfuscator Tool,” https://javascriptobfuscator.herokuapp.com/, 2018.

[5] Yosuke HASEGAWA, “jjencode - Encode any JavaScript program using only symbols,” http://utf-8.jp/public/jjencode.html, 2018.

[6] 高森健太郎, 岩本舞, 小島俊輔, 中嶋卓雄, “マハラノビス距離を用いた難読化マルウェア JavaScript の検出, ” Vol.2014-DPS-161, No.17, pp.1-7, 2014.

[7] 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～,” Computer Security Symposium 2013, 2013.

[8] Alexa Internet, “Inc., Alexa Top 500 Global Sites,” https://www.alexa.com/topsites, 2018.

[9] 平井有三, “はじめてのパターン認識,” 森北出版株式会社, 2012.