jjencode で難読化された JavaScript の検知

1180357　中村 弘亮　【 セキュリティシステム研究室 】

1　はじめに

JavaScript の難読化は，主にそのコード自体の目的を隠したり，内部的な動作を分かりづらくするために使われる．近年では，難読化した JavaScript を埋め込み，そのサイトを訪れた端末にマルウェアをダウンロードさせる Drive-by-Download 攻撃も発生している．ただし難読化は悪意のあるコードに施されるだけでなく，真似されたくないアルゴリズムを読めなくしたり，特定の環境でしか動作しないコードごと難読化しコピー防止のため使われる可能性もある．既存の研究は難読化された悪意のあるコードと一般的なコードを判別するというものであり，一般的なコードが難読化された場合のことは考慮されていない [1]．そこで本稿では難読化された一般的なコードと難読化されていないコードを分類する方法を提案する．さらに，図 1 に示すように JavaScript 内にアルファベットや数字が登場しない，記号のみの難読化コードを出力できる jjencode[2] といった難読化手法なども存在しているため，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ることを示す．

```
$=~[];$={___:++$,$$$$:(![]+"")[$],__$:++$,$_$_:(![]+"")[$],_$_:++
)[$],_$$:++$,$$$_:(!""+"")[$],$__:++$,$_$:++$,$$__:({}+"")[$],$$_
;$.$_=($.$_=$+"")[$.$_$]+($._$=$.$_[$.__$])+($.$$=($.$+"")[$.__$]
.$$_])+($.$=(!""+"")[$.__$])+($._=(!""+"")[$._$_])+$.$_[$.$_$]+$.
._$$]+$.__+$._+$.$+$.$$;$.$=($.___)[$.$_][$.$_];$.$($.$($.$$+"\""
$.__$+$._$_+"\\"+$.__$+$._$_+$._$$+"\\"+$.$__+$.___+"*/\\"+$.$__+
$_$_+"\\"+$.__$+$.$$_+$.___+"\\"+$.__$+$.$_$+$.__$+"."+(![]+"")[$
+$.$$_$+"_"+$.___+"("+$.$$$$+$._+"\\"+$.__$+$.$_$+$.$$_+$.$$__+$.
"\\"+$.__$+$.$_$+$.$$_+"(_){\\"+$.__$+$.$$_+$.$$_+$.$_$_+"\\"+$._
+"\\"+$.__$+$.$$_+$.$$$+"\\"+$.__$+$.$_$+$.__$+"\\"+$.__$+$.$_$+$
$_+$.$$$+"="+$.__+"\\"+$.__$+$.$_$+$.___+"\\"+$.__$+$.$_$+$.__$+"
_$+$._$_+"\\"+$.__$+$.$$_+$.$$_+$.$_$_+"\\"+$.__$+$.$$_+$._$_+"\\
+$.$$$$+$.$_$_+",\\"+$.__$+$.$_$+$.___+$.$_$_+",\\"+$.__$+$.$_$+$
.___+$.$_$_+","+(![]+"")[$._$_]+$.$_$_+",\\"+$.__$+$.$$_+$._$_+$.
```

図 1　jjencode で難読化を施したコードの例

2　提案方法

本研究では，JavaScript を文字出現頻度の違いを学習させた SVM を用いて難読化した一般的な JavaScript と難読化していない一般的な JavaScript に分類する．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で難読化自体が判別可能かを検証できる．また，難読化に jjencode を利用することにより，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可能であることを示す．

3　評価実験

一般的な JavaScript として，Alexa[3] が公開しているアクセス数トップ 500 のうち 10 位以内のサイトから計 438 個の JavaScript を取り出した．各サイトのトップページで使用されていた JavaScript は html ファイル内に直接スクリプトタグとして書かれているものと，スクリプトタグ内から url により参照されているものがあり，その両方を取得している．表 1 に取得した JavaScript のデータについて示した．

表 1　一般的な JavaScirpt のデータ

合計	14,754,808 bites
最大	1,682,739 bites
最小	23 bites

文字種に関しては ASCII の 0x21 から 0x7e までの 94 文字のみとし，それらの文字数の総数からそれぞれの文字が使用された回数を割ることによって各文字の出現頻度を求めた．ここで，各文字の出現頻度は合計で 1 になることから，正規化は行っていない．438 個のスクリプトのうち半数の 219 個に jjencode を施し，SVM を用いて機械学習を行った．また，交差検定は 10 分割で行った．

4　実験結果

RBF(ガウス) カーネル，γ=1.0, C=1.0 のときに正答率が 100%だった．また，既存の方法だと 98.84%だったことから，jjencode によって難読化した一般的なコードは難読化していない一般的なコードと比べ特徴が大きいことが考えられる．

5　まとめ

jjencode は記号のみで難読化を施すため，その特徴的な難読化の仕方から正答率 100%で判別できたと考えられる．また，難読化後のコードが記号のみとなる場合でも難読化を検知することが可能であったことから jjencode の亜種などが出てきたとしても，同じような方法を用いて検出できる可能性があることもわかった．

参考文献

[1] 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, pp.1-7, 2014.

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

[3] Alexa Top 500 Global Sites, Alexa (online), “https://www.alexa.com/topsites”, 2016.