6W-4

不定回符号化に基づく JavaScript 言語の難読化についてとその評価

小澤 俊介 † 大囿 忠親 ‡ 新谷 虎松 ‡
名古屋工業大学知能情報システム学科 † 名古屋工業大学大学院工学研究科情報工学専攻 ‡
e-mail: {kozawa, ozono, tora}@ics.nitech.ac.jp

1 はじめに

近年，WWW において発信される情報の高度化に伴い，WWW 上で公開されるデジタルコンテンツの保護意識が高まっている．本稿では，Web コンテンツにおいて重要な役割を担っている JavaScript に注目する．JavaScript プログラムはサーバからダウンロードされ，ローカルにおいて実行される．そのため，JavaScript プログラムの流出を防止することは困難である．

本稿では プログラムのアルゴリズム等の保護を目的とする．従来のソフトウェアプロテクションの技術 [1] を用いたとしても，JavaScript 言語はテキストベースのインタプリタ型言語であるため，復号のアルゴリズム，鍵，暗号文を完全に隠蔽することができない．そのため，容易に平文を得られてしまう．本稿では，プログラムに対して難読化処理を行い，プログラムの解析を困難にすることでアルゴリズム等の保護を可能にする．

2 プログラム置換による難読化

プログラムの解析を困難にするため，プログラム置換による難読化処理を行う．プログラム置換とは，プログラムのある行とある行を入れ替えることである．プログラム置換によってプログラムの実行される順序を入れ替えることでプログラムの難読化を行う．

2.1 置換アルゴリズム

図 1 に置換アルゴリズムを示し，以下で説明する．

n 行のプログラムに対して難読化を行うとする．まず，サイズ m の置換テーブル T を作成し，置換回数 k $(< m)$ をランダムで決定する．次に，置換を行う行番号 i, j $(< n$, $i \neq j)$ をランダムに決定し，プログラムの i 行目と j 行目の入れ替えを行う．そして，置換を行った行番号 i, j を "$i - j$" として，置換テーブルに格納する．つまり，l 番目の置換テーブルは $T_l = i - j$ となる．この処理を k 回繰返す．k 回置換処理を行った後，置換テーブル T の要素を逆順にする．つまり，$T = [T_k, T_{k-1}, ..., T_2, T_1]$ となる．その後，再び i, j $(< n$, $i \neq j)$ をランダムに決定し，"$i - j$" として，置換テーブルに格納する．しかし，ここではプログラムの入れ替えは行わない．ここで生成される置換テーブルの要素はダミーとなる．このダミーを生成する処理を $m - k + 1$ 回繰返す．これにより，順序がバラバラになったプログラムが生成される．

2.2 復元アルゴリズム

難読化されたプログラムは実行不能な状態になっているため，実行するには復元する必要がある．以下で復元アルゴリズムの説明をする．

A method and evaluation for obfuscation of JavaScript language based on indeterminate number of encoding

Shunsuke KOZAWA, Tadachika OZONO, and Toramatsu SHINTANI

Dept. of Intelligence and Computer Science, Nagoya Institute of Technology, Gokiso, Showa-ku, Nagoya, 466-8555 JAPAN

```
1 : Input: Program  P = {P_1, P_2, ..., P_n}
2 : Output: Obfuscated Program
              P' = {P'_1, P'_2, ..., P'_n}
3 : procedure Encode (P)
4 : begin
5 :     P' ← P
6 :     m ← Table T's size
7 :     String[] : T
8 :     k ← random Integer (<m)
9 :     for t ← 0 to k do
10:         i , j ← random Integer (<n , i ≠ j)
11:         Substitute P'_i for P'_j
12:         T_t ← i−j
13:     end for
14:     Reverse table T
15:     for t ← k to m do
16:         i , j ← random Integer (<n , i ≠ j)
17:         T_t ← i−j
18:     end for
19:     return P'
20: end.
```

図 1: 置換アルゴリズム

復元は置換テーブルに基づいて難読化されたプログラムの置換を行う．置換テーブルから置換を行う行番号を取得し，行の入れ替えを行い，プログラム実行する．この処理を置換テーブルのサイズ m 回行う．ただし，正常にプログラムが実行されるのは k 回のときだけで，それ以外ではプログラムは動作しない．

2.3 置換回数の隠蔽

復元するとき，プログラムを実行する置換回数である k の値が必要となる．この k が難読化されたプログラムを復元するプログラム内に平文で記述されていた場合，容易にプログラムを解析されてしまう．そのため，値 k を隠蔽し，解析を困難にする必要がある．本稿では，値 k を隠蔽する手法として HTML 文書へのステガノグラフィ手法を用いる．

HTML や XML のような構造化文書は，文書の内容そのもの以外に，文書の論理的な構造に関する情報や体裁に関する情報を表現する部分を持っている．文献 [2] では，XML が構造化文書であることを利用して XML へのステガノグラフィ手法の提案をしている．HTML も構造化文書である点は XML と同様である．そこで，HTML の表記の揺らぎを利用して置換回数 k を隠蔽する．

HTML4.01 勧告 [3] によると，表記上はタグを閉じるブランケットの前にいくつかの空白を配置でき，これらの空白は HTML 文書を解釈するときには無視される．このタグ中の空白を利用して，タグを閉じるブランケットの前の空白の有無でタグ 1 つごとに対して 1 ビットの情報を埋め込むことができる．

```
<tag> , </tag>   : 0
<tag > , </tag > : 1
```

上記の手法を利用して HTML 文書へ置換回数を隠蔽する．置換回数 k を 2 進数表記にし，隠蔽を行う．

2.4 JavaScript の解読

JavaScript の解析は，プログラムをローカルにダウンロードし，ローカルでの動的解析がよく行われる．そのため，デバッガや alert 攻撃などを用いて変数の内容を監視することができる．alert 攻撃とは，alert 関数や write 関数などを用いて変数の内容を表示させることである．

JavaScript では文字列を引数とし，引数の文字列を JavaScript コードとして実行する eval 関数がある．通常，暗号化されたプログラムを復元し，実行するときにはこの eval 関数が利用される．しかし，復元された JavaScript コードを実行する eval 関数自体を秘匿することは不可能である．そのため，eval 関数の引数となっている変数が復元された JavaScript プログラムだと推測され，変数の内容を表示させることで容易に復元されたプログラムが得られてしまう．

本稿で提案する手法では，復元の際，プログラム置換が行われる毎にプログラムを実行する．そのため，プログラムの解析者が デバッガや alert 攻撃を用いて復元されたプログラムを得ようとしても最大置換テーブルのサイズである m 回プログラムを解析し，正しいプログラムかどうかを判断しなければならないことになる．複数回プログラムを人手で解析するのは困難であるため，デバッガや alert 攻撃によるプログラムの解析を防ぐことができる．

3 評価実験

提案した手法により難読化したプログラムと難読化していないプログラムの解析を試みてもらう実験を行った．

3.1 実験方法

実験を行うために以下のプログラムを用意した．

- 難読化していない約 50 行のプログラム A, B, C
- A, B, C に対して難読化したプログラム A', B', C'

実験に用いる難読化していないプログラム A, B ,C は JavaScript の知識をある程度知っていれば理解できる簡単なプログラムである．また，難読化したプログラム A', B', C' の置換テーブルのサイズ m は 300 である．

被験者は，同研究室の学部生 6 名と院生 7 名の計 13 名である．各被験者に A, B, C の中から 1 つ，A', B', C' の中から 1 つの計 2 つのプログラムの解析を試みてもらった．被験者へのプログラムの割り当ては，難読化していないプログラムと難読化したプログラムが同じにならないように A と B'，B と C'，C と A' の組み合わせにして割り当てた．

被験者にはまず，こちらで指定した URL へアクセスしてもらい，そこから解析を始めてもらった．指定した URL は難読化していないプログラムあるいは難読化したプログラムを外部ファイルとして読み込む HTML への URL である．この HTML には JavaScript プログラムを読み込むための script タグが記述されている以外，特別な記述はなされていない．解析の順番は難読化していないプログラム，難読化したプログラムの順に行ってもらった．正しく解析できたかを調べるために，プログラムの内容やプログラムに含まれる変数の役割などの質問を行った．

3.2 実験結果

表 1 に難読化していないプログラムの解析に要した時間と難読化したプログラムの解析に要した時間，難読化したプログラムの解析に成功したかどうかを示す．

難読化していないプログラムは被験者全員が解析に成功し，解析の平均所要時間は 13 分であった．これに対し，難読化したプログラムの解析に成功したのは 13 名中 2 名であった．

表 1: 実験結果

被験者	難読化していないプログラムの解析時間 (分)	難読化したプログラムの解析時間 (分)	解析成否
A	12	65	×
B	26	180	○
C	8	60	×
D	18	201	○
E	5	60	×
F	20	120	×
G	15	60	×
H	8	60	×
I	7	60	×
J	5	90	×
K	10	50	×
L	5	40	×
M	30	60	×

表 2: 置換回数による実行速度 (ミリ秒)

	10 回	30 回	50 回	100 回	200 回	300 回
A'	17	34	46	84	149	231
B'	21	41	64	151	229	299
C'	19	57	89	135	181	276

まず，解析に成功した 2 名について考える．解析を成功した被験者 B と D が難読化していないプログラムの解析に要した時間 20 分程度である．これに対し，難読化したプログラムの解析に要した時間は 3 時間以上である．このことから難読化していないプログラムの解析コストに対して，難読化したプログラムの解析コストは 9 倍以上となることがわかる．

次に，解析を断念した 11 名について考える．この 11 名の難読化したプログラムの解析に要した平均時間は約 60 分である．解析を断念した理由として更なる解析コストの増加が考えられる．

解析を成功した 2 名，解析を断念した 11 名に関して，共通に言えることとして解析コストの増大が挙げられる．よって，本稿で提案する難読化手法が有効であることがわかる．

実行速度は難読化前は 5 ミリ秒以下であるのに対し，難読化後は置換回数により表 2 のようになった．置換回数により実行速度が遅くなってはいるが，300 回置換しても 0.3 秒未満であるので，ブラウジングの弊害にはならない．そのため，提案する手法は実用に耐えうると考えられる．

4 おわりに

本稿では，JavaScript 言語のためのプログラム置換に基づく難読化手法について述べた．この難読化手法を用いることで，alert 攻撃による解析の防止ができ，プログラムの解析を困難にすることができる．これにより，プログラムのアルゴリズムなどの保護が可能となる．

参考文献

[1] M Madou, B Anckaert, P Moseley, S Debray, B De Sutter, and K De Bosschere, "Software protection through dynamic code mutation", Proceedings of the 6th International Workshop on Information, 2005.

[2] 井上信吾，村瀬一郎，滝澤修，松本勉，中川裕志，"XML におけるステガノグラフィ手法の提案"，電子情報通信学会 暗号と情報セキュリティシンポジウム (SCIS2002), pp.301-306, Jan. 2002.

[3] HTML 4.01 Specification http://www.w3.org/TR/1999/REC-html401-19991224/