情報処理学会論文誌 Vol.61 No.6 1134–1145 (June 2020)

推薦論文

Proxyオブジェクトを用いた解析妨害JavaScriptコード解析支援システムの実現

上川 先之[1,2] 秋山 満昭[2,a)] 山内 利宏[1,b)]

受付日 2019年7月12日, 採録日 2020年3月13日

概要：Web を介するサイバー攻撃に使用される JavaScript コードは，攻撃コードの意図の隠蔽や検知回避のために，難読化やクローキングなどの解析妨害が施されていることがある．このような解析妨害コードの解析が課題であり，迅速に解析できる手法やシステムが求められる．そこで，我々は解析妨害 JavaScript コード解析支援システムを提案する．提案システムは，JavaScript コードによるブラウザ API 操作を網羅的に捕捉し，API 操作ログを出力することにより，JavaScript コードの挙動を把握できるようにする．API 操作の捕捉に Proxy オブジェクトを利用することで，既存手法では捕捉しきれない API 操作を捕捉する．また，変数の参照を置き換えることにより，置き換え禁止の API に対する API 操作を捕捉可能にする．本論文では，提案システムのコンセプトと，コンセプトを実現する解析手法の実現方式について述べる．また，提案システムの評価として，解析妨害 JavaScript コードの解析を行った結果を報告する．

キーワード：難読化 JavaScript，解析支援システム，動的解析

Support System for Assessing Anti-analysis JavaScript Code by Using Proxy Objects

Hiroyuki Uekawa[1,2] Mitsuaki Akiyama[2,a)] Toshihiro Yamauchi[1,b)]

Received: July 12, 2019, Accepted: March 13, 2020

Abstract: JavaScript code used by web-based attacks is usually protected by some anti-analysis techniques such as obfuscation or cloaking in order to hide its intent or avoid detection. Analyzing such code becomes an urgent task to counter cyber attacks. Therefore, we propose an analysis support system for anti-analysis JavaScript code. The proposed system comprehensively monitors browser API operations and outputs API operation logs for helping analyst's understanding the behavior of code. By using Proxy objects to capture API operations, the proposed system successfuly monitors API operations that could not be captured completely by existing methods. In addition, by replacing variable references, it is able to comprementally monitor API operations for non-replaceable APIs. In this paper, we describe the concept of the proposed system and the implementation of analysis method. We also report the result of analyzing anti-analysis JavaScript codes as an evaluation.

Keywords: obfuscated JavaScript, analysis support system, dynamic analysis

1. はじめに

Web においては，JavaScript を使用する攻撃が多く存在する [1]．スクリプト言語である JavaScript は，Web ペー

[1] 岡山大学大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University, Okayama 700–8530, Japan
[2] NTT セキュアプラットフォーム研究所
NTT Secure Platform Laboratories, Musashino, Tokyo 180–8585, Japan
[a)] akiyama@ieee.org
[b)] yamauchi@cs.okayama-u.ac.jp

本論文の内容は 2017 年 10 月のコンピュータセキュリティシンポジウム 2017（CSS2017）で報告され，同プログラム委員長により情報処理学会論文誌ジャーナルへの掲載が推薦された論文である．

ジに埋め込むことで，ページ閲覧者の Web ブラウザを制御することができる．攻撃者はこの仕組みを悪用し，悪意のある JavaScript コードを埋め込んだページをユーザに訪問させることで攻撃を行う．JavaScript を使用する攻撃として，Web アプリケーションの脆弱性を悪用するクロスサイトスクリプティング（XSS）攻撃や，Web ブラウザの脆弱性を悪用するドライブバイダウンロード（DBD）攻撃がある [2]．たとえば，DBD 攻撃では，Web ブラウザの脆弱性を悪用してマルウェアを秘密裏にダウンロードさせるために，多くの場合 JavaScript が使用される [3]．また，XSS 攻撃や DBD 攻撃では，本来の攻撃を行う悪性な Web サイトで JavaScript が使用されるだけでなく，一般の Web サイトが改ざんされて，悪性な JavaScript コードを埋め込まれる場合がある [4], [5]．

サイバー攻撃の対策を行うには，攻撃コードを解析し，どのような攻撃なのかを明らかにすることが重要である．具体的には，標的となる環境や悪用する脆弱性を特定することである．しかし，Web を介するサイバー攻撃に使用される JavaScript コードは，解析妨害が施されていることがあり [5]，特にエクスプロイトキットで生成された悪性 Web サイトに含まれる JavaScript コードには，解析妨害が施されている場合が多い [6]．解析妨害を行う JavaScript コード（以降，解析妨害 JavaScript コード）では，プログラムを理解が困難な等価なプログラムに変換する難読化や，クライアントの環境によって動作を変えるクローキングの 2 種類の方法が用いられる．JavaScript コードに解析妨害が施されている場合，手動での解析が難しくなり，解析にかかる時間が長くなる．よって，迅速な対応が求められるサイバー攻撃において，特に解析妨害 JavaScript コードを解析する手法やシステムが求められている．

既存の JavaScript コードの解析手法として，特定の API をフックして情報を得る手法と，API のエミュレーションにより Web ブラウザの環境を擬似的に再現する手法がある．しかし，これらの手法には，2.3 節で後述するように課題がある．そこで，解析妨害 JavaScript コード解析支援システムを提案し，既存の解析手法の課題に対処する．本論文では，本システムのコンセプト，コンセプトを実現する解析手法の実現方式，および提案システムの評価について述べる．

本研究の貢献は，以下に示すとおりである．

(1) ブラウザ API の網羅的な捕捉を実現するために，Proxy オブジェクトを利用した新たな解析手法を提案する．これにより，既存の解析手法における課題へ対処する．著者らの調べた限りでは，Proxy オブジェクトを JavaScript コードの解析に用いた手法は，まだ提案されていない．
(2) 一部のブラウザ API への Proxy オブジェクト適用のために，変数の参照を置き換える手法を提案する．これにより，従来の手法ではフックできなかった置き換え禁止のブラウザ API に対して，Proxy オブジェクトを適用できる．
(3) 提案する解析手法を解析妨害 JavaScript コード解析支援システムとして実現する．本システムは，さらなる解析に有用な情報を解析者に提示することで，解析者による解析を支援する．また，事前の手動での解析を必要とせず，自動で動的解析を行うことができる．
(4) 提案システムの有用性を評価するために，難読化された JavaScript コードの解析，およびクローキングを行う JavaScript コードの解析を行い，解析結果について考察した．難読化解析においては，提案システムにより，ブラウザ API 操作ログが取得できることを示した．クローキング解析においては，過半数の場合で問題なく解析できることを確かめ，また，提案システムにおけるいくつかの課題を明らかにした．

2. JavaScript コードにおける解析妨害技術と解析手法

2.1 解析妨害技術

2.1.1 難読化

難読化とは，あるプログラムを理解が困難な等価なプログラムに変換することである [7]．一般に，難読化は，プログラムを不正な解析から保護するために行われる [7]．プログラムの解析を困難にすることで，プログラムの改ざんやプログラムコードの剽窃を防ぐことができる．

一方，サイバー攻撃に用いられる攻撃プログラムは，攻撃者によって難読化される場合がある．攻撃者は，難読化によって攻撃プログラムの動作を解析されにくくし [5], [8], [9]，その攻撃プログラムの目的を隠蔽する [10], [11]．また，ウイルス対策ソフトウェアやファイアウォールによる検知を回避するためにも難読化が行われる [5], [11], [12]．難読化により等価なプログラムが生成されることから，多様な亜種の作成を容易にすることも目的としてあげられる [8]．

攻撃に使用される JavaScript コードも例外ではない [5], [10]．DBD 攻撃の例では，リダイレクト先である悪性な Web サイトの URL の隠蔽や，リダイレクトすること自体の隠蔽のために，JavaScript コードが難読化される．

難読化は，大きく以下の 3 種類に分けられる [13]．

(1) レイアウト難読化
識別子名，コメント，およびインデントなどのプログラムの実行に不要な情報を置き換える変換である．

(2) データ難読化
データ構造やデータ表現を変える変換である．たとえば，文字列のエスケープや，数値をそれと同等な演算式に置き換える変換があげられる．

(3) 制御フロー難読化
制御フローを変える変換である．たとえば，無駄な

```
eval(unescape("%6c%6f%63%61%74%69%6f%6e%2e%68%72%65%66%3d%22
%68%74%74%70%3a%2f%2f%77%77%77%2e%65%78%61%6d%70%6c%65%2e%63
%6f%6d%2f%22%3b"))
```

(a) エンコードされたコード，デコーダ，およびコード実行部

```
location.href="http://www.example.com/";
```

(b) デコードされた JavaScript コード

図 1 難読化 JavaScript コードの例
Fig. 1 Example of obfuscated JavaScript code.

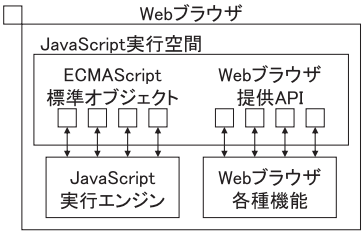


図 2 Web ブラウザにおける JavaScript の API
Fig. 2 APIs of JavaScript in a web browser.

ループ文の挿入や，関数のインライン展開があげられる．

JavaScript コードにおいては，特に，コードを圧縮するためにレイアウト難読化が施されることが多い．また，JavaScript には，文字列を JavaScript コードとして実行することのできる `eval` 関数があり，容易にコード全体を難読化できることから，`eval` 関数に渡される文字列にデータ難読化が施されることも多い．

2.1.2 クローキング

攻撃プログラムが，クライアントの環境によって動作を変えることがある．このことをクローキングと呼ぶ．たとえば，Web ブラウザが特定の脆弱性を持つバージョンである場合のみ攻撃を行い，そうでない場合は何もしないといったように，動作を変える攻撃が存在する．クローキングには，クライアントへの応答を変えるサーバ側クローキングと，ブラウザ側でプログラムの処理流れを変えるクライアント側クローキングの 2 つがある．クローキングを行う攻撃プログラムでは，Web ブラウザのバージョンや使用しているプラグインの情報などのクライアントの環境情報を取得する．このことをフィンガープリンティングと呼ぶ．クライアント側クローキングにおけるフィンガープリンティングでは，JavaScript を用いて，Web ブラウザから提供される環境情報を取得する．

クローキングを行う攻撃プログラムの動的解析を行う場合，通常は，攻撃プログラムが対象としている環境でなければ攻撃が再現しない．このため，攻撃プログラムが対象としている環境で攻撃プログラムを実行するか，または何らかの方法で攻撃を行わせるように細工を施す必要がある．このように，攻撃プログラムがクローキングを行う場合，攻撃を実行する条件を解析する手間が増えるため，クローキングは解析妨害の一種と見なせる．

2.2 関連研究

既存の JavaScript コード解析手法として，API Hooking とブラウザ API のエミュレーションを説明する．

2.2.1 API Hooking

難読化 JavaScript コードは，エンコードされた JavaScript コード，デコーダ，およびコード実行部の 3 つからなるものが多く見られる．**図 1** の例では，URL エスケープされた JavaScript コードを `unescape` 関数でデコードし，得られた (b) の JavaScript コードを `eval` 関数で実行している．このような難読化 JavaScript コードでは，デコードした JavaScript コードをコード実行部で文字列として `eval` などの関数に渡す．このため，どれほど複雑にエンコードされていても，`eval` などの関数をフックすることで，デコードされた JavaScript コードを得ることができる．このように，特定の関数をフックする手法を API Hooking [5], [14] と呼ぶ．

文献 [5] では，コードを直接実行できる `eval` 関数や `setTimeout` 関数のほかに，間接的にコードを実行できる `document.write` 関数など 10 種類の API をあげ，これらをフックすることで，デコードされた JavaScript コードの取得を実現している．さらに，フィンガープリンティングやリダイレクトに用いられる API をフックすることで，フィンガープリンティングの検出と環境偽装，およびリダイレクト先 URL の抽出を実現している．

2.2.2 ブラウザ API のエミュレーション

JavaScript コードの解析ツールとして，jsunpack-n [15] と JSDetox [16] がある．これらのツールは，Web ブラウザを用いずに JavaScript コードの動的解析を行っている．

Web ブラウザにおける JavaScript の API について説明する．Web ブラウザにおける JavaScript では，ECMAScript 標準オブジェクト（以降，標準 API）と Web ブラウザ提供 API（以降，ブラウザ API）の 2 種類の API が，組み込みオブジェクトとして提供される（**図 2**）．標準 API は，言語仕様を実現するための API であり，ブラウザ API は，Web ブラウザを制御するための API である．たとえば，標準 API として，配列を扱うための `Array` や文字列を扱うための `String` などがある．一方，ブラウザ API には，JavaScript から HTTP 通信を行うための `XMLHttpRequest` や HTML 要素の操作を可能にする `HTMLElement` などがある．

上記解析ツールは，動的解析のために，JavaScript 実行エンジンを利用している．このため，JavaScript 実行空間には，Web ブラウザと同様に標準 API が提供される．しかし，JavaScript 実行エンジンは Web ブラウザの機能を持たないため，ブラウザ API は提供されない．そこで，上記解析ツールでは，一部のブラウザ API を独自に定義し，ブラウザ API のエミュレーションを行う．

ブラウザ API のエミュレーションでは，解析者が自由に API を定義できる．このため，たとえば，ブラウザ API により提供される環境情報を自由に変えることができ，フィンガープリンティングを行う JavaScript プログラムの動的解析に有効である．

2.3 課題

解析妨害が施された JavaScript コードに対する既存の解析手法には，以下の課題がある．

（課題 1）コード実行 API を使用しない種類の難読化 JavaScript コードの解析

コード実行 API を使用する種類の難読化 JavaScript コードについては，API Hooking により，デコードされた JavaScript コードを容易に取得できる．しかし，コード実行 API を使用しない種類の難読化（主にレイアウト難読化）が施される例が存在する．このような難読化 JavaScript コードについては，既存手法では，より可読性の高いコードを得られない．このため，コード実行 API を使用する種類の難読化と比べて，解析が難しい．

（課題 2）解析者の想定していない API への操作の捕捉

API Hooking やブラウザ API のエミュレーションでは，解析者の想定していない API への操作を捕捉できない．たとえば，解析者が環境情報として Web ブラウザ名と言語設定の 2 つを想定し，API をフックまたはエミュレートしている場合，プラグイン情報の取得操作が行われても，そのことに気づかない可能性がある．このことは，従来と異なる環境情報を取得する JavaScript コードを解析する際に問題となる．

3. 解析妨害 JavaScript コード解析支援システム

3.1 目的と要件

本論文では，解析妨害 JavaScript コード解析支援システム（以降，提案システム）を提案する．提案の目的は，2.3 節で述べた既存の解析手法における課題への対処である．また，解析に有用な情報を提示し，解析者による解析を支援することも目的の 1 つである．

前述の目的を達成するために，以下の要件が提案システムに求められる．（要件 1）と（要件 2）は，それぞれ（課題 1）と（課題 2）への対処である．

（要件 1）可読化の難しいコードの解析

難読化 JavaScript コードに対し，より可読性の高いコードが容易に得られない場合においても，解析に有用な情報が得られることが望ましい．たとえば，フィンガープリンティングに関する情報は，攻撃のターゲットとなる環境の推測に有用である．また，SWF ファイルの読み込みやリダイレクトなどの挙動が分かれば，攻撃手法やキャンペーンの推測につながる．これらの情報を得られれば，より精密な解析が可能になる．

（要件 2）解析者が想定していない API の操作の捕捉

解析者が想定していない API の操作を捕捉できるようにすることで，API 操作に関する幅広い情報を得る．これにより，たとえば，未知のフィンガープリンティングが行われ，想定していなかった API へのアクセスがあったことに気づくことができるようになる．

（要件 1）と（要件 2）を満たす解析手法のコンセプトとして，ブラウザ API 操作の網羅的な捕捉を考えた．提案システムでは，このコンセプトによる解析を可能とする．このコンセプトおよびそれを実現する解析手法について，3.2 節で説明する．

なお，ブラウザ API のエミュレーションによる動的解析では，ブラウザ API のエミュレーションが不完全である可能性があり，本来の Web ブラウザでの挙動が再現できない可能性がある．このため，実際の Web ブラウザを用いて解析できることが望ましい．提案システムでは，実環境で JavaScript コードを正確に実行するため，実際の Web ブラウザを用いた動的解析を可能とする．これにより，目的とする Web ブラウザとそのバージョンの環境において，詳細な情報を取得できる．さらに，様々な Web ブラウザで解析を可能とするため，Web ブラウザおよび JavaScript 実行エンジンの改変を必要としない解析手法を実現する．

以上の方針に基づき，解析妨害 JavaScript コードを自動で動的解析するシステムを実現する．

3.2 ブラウザ API 操作の網羅的な捕捉

Web ブラウザにおける JavaScript には，コンソールプログラムにあるような標準入出力がない．また，イベント駆動型プログラミングを想定した設計になっている．このため，何らかの目的を持った JavaScript プログラムは，Web ブラウザから何らかのイベントによる入力を受け，その実行結果を Web ブラウザに反映する．つまり，JavaScript プログラムは，入出力インタフェース相当のブラウザ API を操作することにより，プログラムの目的を達成する．このことをふまえ，ブラウザ API の操作を捕捉し，そのログを取得する．この方法であれば，コードを読むことなく，JavaScript プログラムの挙動の把握でき，また解析に有用な情報を得ることができる．以上のコンセプトを実現する次の解析手法により，（要件 1）を満たす．

ブラウザ API 操作を網羅的に捕捉するために，標準 API である Proxy オブジェクトを用いた手法（以降，Proxy 手法）を提案する．この手法のアイデアは，組み込みオブジェクトであるブラウザ API への参照をすべて Proxy オブジェクトを指すように差し替えることにより，ブラウザ API に対する操作をすべて Proxy オブジェクトが仲介す

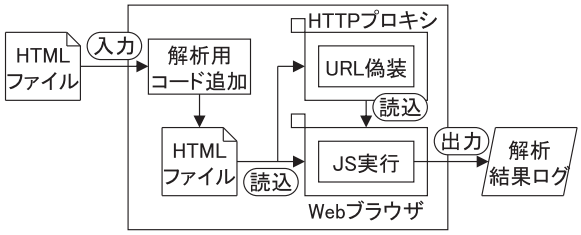


図 3 提案システムの全体図
Fig. 3 Overview of proposed system.

ることである（詳細は 4.2 節で説明）．この手法を用いることで，ブラウザ API に対するすべてのプロパティ参照とメソッド呼び出しを漏れなく捕捉できる．また，捕捉した内容をログに出力することで，対象の JavaScript コードが何を実行したのかを解析することを支援できる．この手法は，API Hooking やブラウザ API のエミュレーションと異なり，網羅的に捕捉するため，解析者の想定していない API に対する操作を捕捉できる．

一部のブラウザ API は，置き換えが禁止されており，単純な再代入による Proxy オブジェクトへの差し替えができない．そこで，変数への再代入と同等の効果を得られる変数の参照置き換え手法（以降，参照置換手法）を提案し，置き換え禁止 API の操作捕捉を可能にする（詳細は 4.4 節で説明）．これにより，グローバル変数へのアクセスや API Hooking で捕捉しきれないブラウザ API の操作などを把握可能となり，より詳細な解析結果を得ることができる．上記の 2 つの解析手法により，（要件 2）を満たす．

3.3 提案システム

3.1 節で述べた方針に基づいて設計した提案システムの全体図を**図 3** に示す．提案システムは，Web ブラウザでの動的解析をベースとしたシステムである．解析者は，解析したい JavaScript コードを含む HTML ファイルを入力し，出力として解析結果のログ（ブラウザ API 操作ログ）を得る．提案システムに HTML ファイルが入力されると，その HTML ファイルに解析のための処理（API 操作捕捉処理など）を行う JavaScript コードが自動で追加される．解析用コードを加えた HTML ファイルを Web ブラウザで読み込み，JavaScript コードを実行することで，Web ブラウザのコンソールに解析結果のログが出力される．

ページ URL をもとにクローキングを行う可能性のある JavaScript コードを解析したい場合がある．このような場合は，URL 偽装の機能を持たせた HTTP プロキシに偽装したいページ URL を与え，偽装したいページ URL にプロキシを介してアクセスすることで解析できる．

提案システムは，ブラウザ API の操作を捕捉し，そのログを取得して解析者に提示するという 3.2 節のコンセプトを実現し，3.1 節で述べた要件をすべて満たす．

Proxy オブジェクトは，JavaScript の仕様を定める ECMAScript において，ECMAScript 2015 [17] から策定された標準 API である．このため，Proxy オブジェクトが実装されている Web ブラウザ（**表 1**）を使用する．

表 1 主要 Web ブラウザにおける Proxy オブジェクトの実装状況
Table 1 Implementation status of Proxy object in major web browsers.

Web ブラウザ	実装バージョン
Internet Explorer	未実装
Microsoft Edge	12 以降
Mozilla Firefox	18 以降
Google Chrome	49 以降

3.4 解析用コード追加処理

図 3 中の解析用コード追加処理の具体的な内容を説明する．入力 HTML ファイルに，Proxy 手法用コードと参照置換手法用コードを挿入する．これらのコードは，3.2 節で説明したブラウザ API 操作の網羅的な捕捉を実現するためのコードである．Proxy 手法を適用するために行う処理は，解析対象の JavaScript コードより前に 200 行程度の JavaScript コードを 1 度挿入するだけである．また，参照置換手法を適用するために行う処理は，解析対象の各 JavaScript コードに対し，それぞれ 50 行程度の JavaScript コードを挿入するだけである．

Proxy 手法および参照置換手法は，解析対象コード毎に挿入するコードを修正する必要がなく，動的解析前に JavaScript コードを解析する必要がない．つまり，これらの解析用コードは，解析対象コードの内容に依存しないため，事前に用意した固定の解析用コードを挿入するだけでよい．以上により，3.2 節で説明したコンセプトによる解析手法を提案システムに実現できる．

4. 実現方式

4.1 ブラウザ API の概要

3.2 節で述べたとおり，提案システムでは，ブラウザ API の操作を捕捉する．JavaScript は，プロトタイプベースオブジェクト指向言語である．C++や Java のようなクラスベースオブジェクト指向言語と異なり，クラスの概念が存在しない．このため，本論文では説明上，特定の操作に関する一連の API 群をクラスと呼ぶ．

ブラウザ API の 1 つである `XMLHttpRequest`（**図 4**）を例に，ブラウザ API の概要および関連する言語仕様について説明する．JavaScript から HTTP 通信を行うためのクラスとして，`XMLHttpRequest`（`XHR` と略記）がある．ブラウザ API として提供されるクラスは，基本的に，そのクラスのインスタンスを生成するためのコンストラクタ[*1]，そのクラスのインスタンスの雛形であるプロトタイプ，お

[*1] コンストラクタやメソッドの実体は関数である．また，関数はオブジェクトとして扱われる．

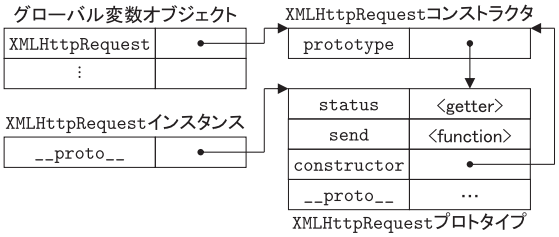


図 4 ブラウザ API の例（XMLHttpRequest）
Fig. 4 Example of browser API (XMLHttpRequest).

```
1  var request = new XMLHttpRequest();
2  request.open("GET", "http://www.example.com/", false);
3  request.send();
4  if (request.status === 200)
5    console.log(request.responseText);
```

図 5 XMLHttpRequest の使用例
Fig. 5 Example of using XMLHttpRequest.

よび各種メソッド[*1]の 3 つから成る．たとえば，XHR クラスは，XHR コンストラクタ，XHR プロトタイプ，およびメソッド（open や send など）から成る．コンストラクタは，prototype プロパティを持ち，そのクラスのプロトタイプを値として持つ．

ここで，XHR を使用する JavaScript コードの例を**図 5** に示す．1 行目で，XHR コンストラクタにより XHR インスタンスを生成し，request 変数に格納している．このとき，まず XHR の名前解決が行われ，グローバル変数オブジェクトから XHR コンストラクタを得る．次に，new 構文により，XHR コンストラクタに対して内部関数 [[Construct]] が呼び出される．この結果，XHR インスタンスとして空のオブジェクトが生成され，その __proto__ プロパティに XHR コンストラクタの prototype プロパティの値である XHR プロトタイプが設定される．

2 行目と 3 行目では，request 変数に格納された XHR インスタンスに対し，open メソッドと send メソッドを呼び出している．このとき，send メソッドの場合，まず XHR インスタンスに対して内部関数 [[Get]] を用いて send の名前解決が行われる．XHR インスタンスは，send プロパティを持っていないため，__proto__ プロパティが指すオブジェクトに対して再帰的に名前解決が行われる．この結果，XHR プロトタイプに send プロパティが見つかり，その send プロパティが持つ関数[*2]を得る．最終的に，得た関数に対して内部関数 [[Apply]] が呼ばれ，メソッドとしてその関数の処理が実行される．4 行目では，同様に内部関数 [[Get]] を用いて status の名前解決が行われ，XHR プロトタイプの status プロパティの値を得る．このとき，status プロパティにはゲッター関数が設定されており，この関数を実行した結果が status プロパティの値として得られる．

[*2] 関数はメソッドとして機能しうる．メソッドの実体である関数だけを見て，それがメソッドとしての関数か否かを判別できない．

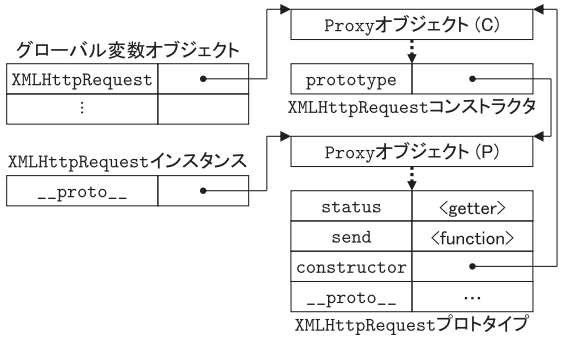


図 6 XMLHttpRequest に Proxy オブジェクトを適用した例
Fig. 6 Application of Proxy object to XMLHttpRequest.

4.2 Proxy オブジェクトによるブラウザ API 操作の捕捉（Proxy 手法）

提案システムでは，Proxy オブジェクトを利用することにより，ブラウザ API 操作の捕捉を実現する．具体的には，ブラウザ API の操作をブラウザ API（オブジェクト）に対する内部関数呼び出しと定義し，これらの内部関数を捕捉する．Proxy オブジェクトは，JavaScript のオブジェクトに対する基本的な操作を内部関数呼び出しレベルで捕捉することができる．

ブラウザ API 操作を捕捉するために，以下を実施し，クラスを構成するコンストラクタ，プロトタイプ，および各種メソッドへのアクセスを，すべて Proxy オブジェクトを介するようにする．

(1) 以下にあげるコンストラクタへの参照を，あらかじめそれぞれの対応する Proxy オブジェクトに差し替える．
  (a) グローバル変数オブジェクトの該当プロパティ
  (b) プロトタイプの constructor プロパティ
(2) 以下にあげるプロトタイプへの参照を，あらかじめそれぞれの対応する Proxy オブジェクトに差し替える．
  (a) コンストラクタの prototype プロパティ
  (b) 他プロトタイプの __proto__ プロパティ
  (c) あらかじめインスタンスとして提供されるオブジェクトの __proto__ プロパティ
(3) Proxy オブジェクトが内部関数呼び出しを捕捉したとき，以下を実施する．
  (a) 対象オブジェクトに対する操作情報として，内部関数の情報を出力する．
  (b) 返却値が関数であれば，その関数に対する Proxy オブジェクトを返却する．
  (c) 返却値が捕捉すべきクラスのインスタンスであれば，そのインスタンスの __proto__ プロパティを Proxy オブジェクトに差し替える．

Proxy オブジェクトを XMLHttpRequest（XHR）に適用した例を**図 6** に示し，図 5 のコードを用いて説明する．まず，1 行目では，XHR の名前解決の結果，XHR コンストラ

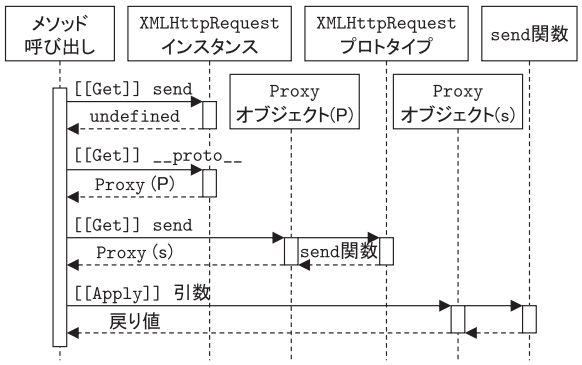


図 7 Proxy オブジェクト適用時の send メソッド呼び出しの流れ
Fig. 7 Sequence of send method to which Proxy object has applied.

クタの代わりに Proxy オブジェクト（C）を得る．これにより，new 構文でインスタンスを生成する際に，Proxy オブジェクト（C）に対して内部関数 [[Construct]] が呼び出される．この Proxy オブジェクト（C）は，呼び出された内部関数の情報をログとして出力し，本来の XHR コンストラクタに対して内部関数を呼び出す．また，生成された XHR インスタンス（request 変数）の __proto__ プロパティには，XHR コンストラクタの prototype プロパティに設定されている Proxy オブジェクト（P）が設定される．

2，3，4，および 5 行目では，プロパティの名前解決のために XHR インスタンスの __proto__ プロパティを参照し，XHR プロトタイプの代わりに Proxy オブジェクト（P）に対して内部関数 [[Get]] を呼び出す．Proxy オブジェクト（P）では，呼び出された内部関数の情報をログとして出力し，本来の XHR プロトタイプに対して内部関数を呼び出す．特に 2，3 行目のメソッド呼び出しでは，内部関数 [[Get]] は，メソッドとしての本来の関数の代わりに，都度新たな Proxy オブジェクトを返す．たとえば，3 行目の send メソッド呼び出しについて，Proxy オブジェクトを適用した際のシーケンス図は**図 7** のようになる．send 関数に対する新たな Proxy オブジェクト（s）に対し，関数実行のために内部関数 [[Apply]] が呼び出され，Proxy オブジェクト（s）は，同様に情報をログとして出力する．以上のようにすることで，ブラウザ API 操作を捕捉し，そのログを取得できるようになる．

4.3 Proxy オブジェクト適用の限界

4.2 節で述べた Proxy 手法には，限界がある．Web ブラウザにより提供される window オブジェクトは，その __proto__ プロパティを書き換えられない．このため，一部のブラウザ API（window.addEventListener メソッドなど）を捕捉できない．この問題への対処として，API Hooking の併用が有効である．

また，インスタンスとして提供されるブラウザ API である window オブジェクトと location オブジェクトは，それぞれのプロトタイプが持たないプロパティを各インスタンスが持つ．window オブジェクトは，グローバル変数オブジェクトとしての役割を持ち，そのプロパティ参照を捕捉することで，グローバル変数への参照を捕捉できる．location オブジェクトは，現在のページ URL に関する情報を提供しており，特にリダイレクトのために利用される．しかし，これらのオブジェクトを指すグローバル変数である window 変数と location 変数は，置き換えが禁止されており，単純な再代入では Proxy オブジェクトに差し替えられない．また，グローバル変数オブジェクトそのものの置き換えもできない．グローバル変数に再代入できない問題は，4.4 節で説明する参照置換手法により対処する．

4.4 変数の参照置き換え（参照置換手法）

置き換え禁止 API の操作捕捉を可能にするため，変数の参照置き換えにより，変数への再代入と同等の効果を得る．

JavaScript における変数の名前解決について説明する．JavaScript の実行コンテキストには，関数の外側のグローバル実行コンテキストと，呼び出された関数ごとの関数実行コンテキストがある．各実行コンテキストは，それぞれ変数オブジェクトを持ち，グローバル変数や関数内変数を管理する．たとえば，グローバルコンテキストで関数 A を呼び，関数 A 内で変数 window が参照されたとする．このとき，window の名前解決のため，現在の実行コンテキストが持つ関数 A の変数オブジェクトから window を探す．関数 A の変数オブジェクトに window がなければ，上位の実行コンテキストであるグローバル実行コンテキストの変数オブジェクト，つまりグローバル変数オブジェクトから探す．

4.3 節で説明したとおり，グローバル変数として提供される window 変数と location 変数は，再代入ができない．これらの変数に Proxy オブジェクトを適用するために，前述した変数の名前解決の仕様を利用する．具体的には，解析対象の JavaScript コードをまるごと 1 つの関数内に入れ，その関数で関数内変数[*3]として window 変数と location 変数を定義する．これにより，関数内でたとえば window 変数が参照された場合，先にその関数の変数オブジェクトから探索されるため，関数内変数の window 変数の値が得られる．この手法で，**図 8** のように，関数 A 内の window 変数と location 変数の値にそれぞれ Proxy オブジェクトを代入する．

Web ブラウザの JavaScript においては，window オブジェクトがグローバル変数オブジェクトとしての役割を担う．このため，通常，window オブジェクトの window プロパティは，グローバル変数 window と同じく window オブジェクト自身を指す．同様に，window オブジェクトの

[*3] いわゆるローカル変数．関数内で var キーワードにより宣言される．

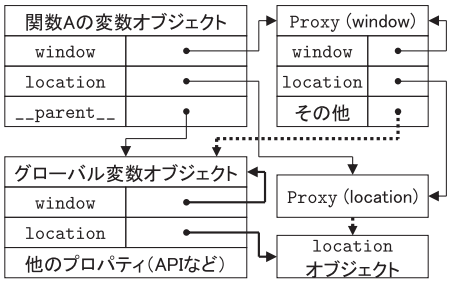


図 8 変数の参照置き換えによる Proxy オブジェクト適用
Fig. 8 Application of Proxy object by replacing references of variables.

location プロパティは，location オブジェクトを指す．このことを考慮し，window オブジェクトに対する Proxy オブジェクトは，window プロパティや location プロパティを参照されたとき，各 Proxy オブジェクトを返すようにする．以上により，window 変数と location 変数が指していた window オブジェクトと location オブジェクトに対する操作を捕捉できるようになる．

本節で述べた参照置換手法には，副作用がある．具体的には，グローバル変数として宣言した変数の性質が変わり，JavaScript コードによってはプログラムの挙動が変わる場合がある．これは，本来グローバル実行コンテキストで実行されるコードが，関数実行コンテキストで実行されることに起因する．このことをふまえ，解析対象の JavaScript コードを修正するか，または本手法を適用せずに解析するなど，場合に応じて対応する必要がある．

また，JavaScript には，window 変数を参照せずにグローバル変数オブジェクトの参照を得る方法が存在する．このため，そのようなコードを含む JavaScript プログラムは，window オブジェクトと location オブジェクトに対する操作を漏れなく捕捉することができない．

5. 評価

提案システムの要件確認，および実用性評価を目的とし，悪性サイトから収集された解析妨害 JavaScript コードを用いて，難読化とクローキングに着目して解析を行う．またその結果に基づいて考察と課題を論じる．

5.1 難読化解析による要件確認

5.1.1 評価方法

提案システムが 3.1 節の要件を満たすことを確認するために，難読化されたコードの挙動を提案システムで解析できるか否かを確認した．評価方法として，実際の攻撃で用いられた解析妨害 JavaScript コードを提案システムに入力し，解析結果について考察する．

JavaScript コードを提案システムで解析する際，可読化の難しいコードでもプログラムの挙動を把握できることを確認するために，JavaScript コードを事前に手動解析しない．また，解析結果について，得られる情報がさらなる解析に有用かどうかに着目して考察する．これにより，（要件 1）を満たすことを示す．また，解析者が明示的に特定の API をフックしなくても，ブラウザ API 操作として解析者が想定していない API のログが得られることを確認する．これにより，（要件 2）を満たすことを示す．

なお，提案システムでは，Proxy 手法と参照置換手法を適用するほか，より解析を容易にするため，標準 API である eval 関数のフックを実施する．また，本評価で用いた Web ブラウザは，Google Chrome 60.0.3112.101（V8 6.0.286）である．

5.1.2 評価結果

解析妨害 JavaScript コードを含む HTML ファイルを D3M データセット [18] のパケットキャプチャデータ[*4]から 1 つ抽出し[*5]，この HTML ファイルを提案システムに入力した．Web ブラウザで動的解析を行い，ブラウザコンソールに出力されたブラウザ API 操作ログを**図 9** に示す．得られたログを見ると，特定の要素 ID の HTML 要素へのアクセス，HTML 要素が持つ情報の取得，および object 要素を含む HTML の書き出しが行われていることを一目で確認できる．ブラウザコンソールを操作してログを詳細表示すると，object 要素で SWF ファイルを読み込もうとしていることも確認できる．また，環境情報として，プラグイン情報（特に Silverlight プラグイン）と User-Agent を取得していることも容易に確認できる．この結果から，攻撃手法やターゲットとなる環境などを推測することができ，さらなる解析に有用な情報が得られているといえる．難読化されたコードをいっさい読むことなくこれらの情報が得られたことから，（要件 1）は満たしているといえる．

別の解析妨害 JavaScript コードとして，Malware-Traffic-Analysis.net [6] から Magnitude EK によるサンプル[*6]を取得し，提案システムに入力した．得られたログを**図 10** に示す．この例では，window[gjrBu1hg(...)] が関数ではないというエラーにより，プログラムが動作を停止している．しかし，エラーメッセージから window[gjrBu1hg(...)] が何であるかを読み取ることができず，本来は，さらなる解析のために JavaScript コードを手動解析する必要がある．そこで，エラーメッセージ直前の提案システムが出力したログを見ると，window.ScriptEngineBuildVersion へのアクセスがあり，その値が未定義であることが分かる．つまり，本評価に用いた Web ブラウザでは，window.ScriptEngineBuildVersion 関数が定義されていないということを推察できる．この情報を元に，この攻撃プログラムがどのような環境を想定しているかを分析したり，独自に関数を定義して攻撃プログラムの挙動を解析

[*4] MD5 ハッシュ値：2563d0cfed67550a5ca940c74d91dd4a
[*5] ファイル名：VF9QAxhUUBkG.html
[*6] ファイル名：2016-07-25-Magnitude-EK-more-html.txt

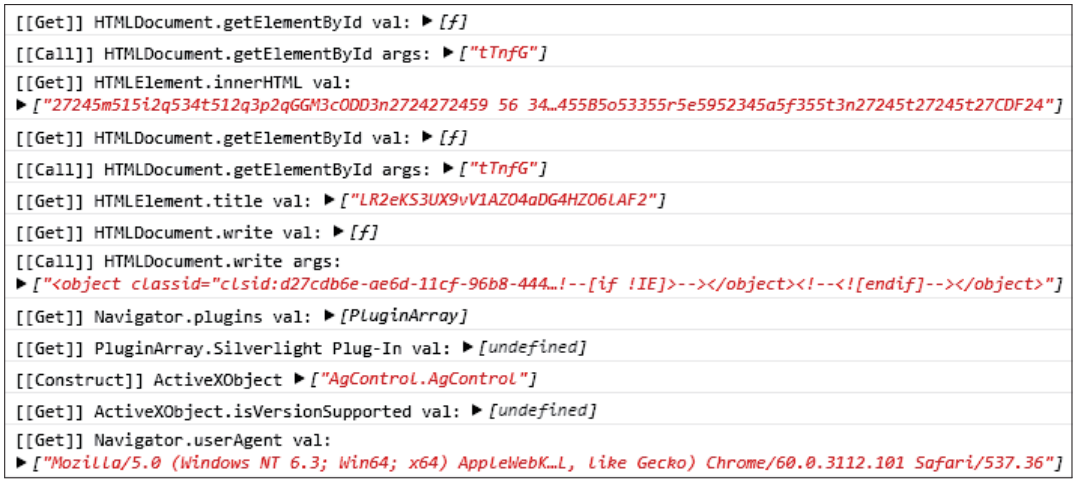


図 9 ブラウザコンソールに出力されたログ（動的解析結果）
Fig. 9 Logs outputted into browser console (result of dynamic analysis).

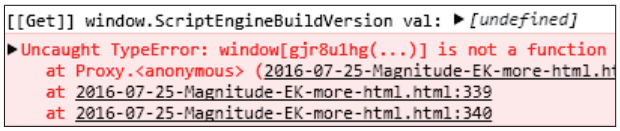


図 10 `window` オブジェクトに対するプロパティ参照のログ
Fig. 10 A log of property access for `window` object.

したりなど，さらなる解析につなげることができる．この例で得られたログは，参照置換手法を適用しなければ得られないログであり，この例では，参照置換手法が有効だったといえる．また，既存手法では，解析者が上記 API（関数）の存在を認識しており，かつ明示的にフックを行わなければ，この API の操作ログを得られない．提案システムでは，解析者が明示的にフックすることなくログを得られたことから，（要件 2）は満たしているといえる．

以上より，提案システムによって，可読化の難しいコードでもプログラムの挙動を把握できることを示した．また，取得された環境情報の種類や未定義のブラウザ API へのアクセスを確認できることから，環境偽装による動的解析などのさらなる解析に有用な情報を得られることを示した．

5.2 クローキング解析による実用性評価

5.2.1 評価方法

提案システムでクローキングの内容を解析できるか否かを確認する．具体的には，クローキングを行う解析妨害 JavaScript コードを解析し，以下の項目について，それぞれ解析できたか否かを確認する．

（項目 1）フィンガープリンティングの有無の確認
解析ログを元に，環境依存の API へのアクセスによるフィンガープリンティングが行われたことを確認できるか否かを確認する．

（項目 2）クローキングの有無の推測
解析ログを元に，フィンガープリンティングの結果を用いて条件分岐などのクローキングを行っていることを推測できるか否かを確認する．

また，実際の攻撃で用いられる様々なエクスプロイトキットによる解析妨害 JavaScript コードを評価に用いることで，提案システムが実用的であることを示す．評価に用いる解析妨害 JavaScript コードとして，Malware-Traffic-Analysis.net [6] で公開されているエクスプロイトキットによる攻撃データから，難読化 JavaScript コードを含むファイルを抽出した．抽出したファイルの一覧を**表 2** に示す．

まず，表 2 の各ファイルを解析した結果を**表 3** に示す．いくつかの事例について，5.2.2 項でケーススタディとして解析結果の詳細を述べる．なお，難読化解析の評価と同様に，eval 関数のフックを実施している．また，本評価で用いた Web ブラウザは，Google Chrome 63.0.3239.84（V8 6.3.292.46）である．

5.2.2 評価結果

5.2.2.1 ケーススタディ：ファイル a27

本ファイルは，KaiXin EK による難読化 JavaScript コードを含む HTML ファイルである．提案システムによる解析の結果，eval 関数のフックおよび Proxy 手法により，eval 関数の実行および script タグの書き出しが行われたことを確認できた．しかし，コード中で定義されるグローバル関数が，参照置換手法の適用によりグローバルでない関数になり，プログラムがエラーにより停止した．

得られた解析ログを元に，デコードされた JavaScript コードを記述した HTML ファイルを作成し，再度解析した．この結果，Proxy 手法により，User-Agent 文字列，言語設定およびプラグイン情報の取得が行われたことを確認できた．

5.2.2.2 ケーススタディ：ファイル b9

本ファイルは，Magnitude EK によるリダイレクトページの HTML ファイルである．`location` オブジェクトを参照し，ページ URL によるクローキングを行っている．し

表 2 Malware-Traffic-Analysis.net から収集した難読化 JavaScript コードを含むファイル
Table 2 List of files which include obfuscated JavaScript code, gathered from Malware-Traffic-Analysis.net.

PCAP ファイル名 ID	EK / キャンペーン	SHA-1 ハッシュ値	パケット番号
2017-11-17-KaiXin-EK-traffic.pcap			
a27	KaiXin EK	e67fdd6ef390f24dcadb0db1f2ac2c35a2b64e2e	6 / 27
a55	KaiXin EK	4ddaf3c85d23f8d3f3238bc8c71445ba0eb4cd39	34 / 55
2017-08-02-Magnitude-EK-sends-Cerber-ransomware.pcap			
b9	magnigate	d028eaa6d6c2ddd0e5d7825ea61205808ef19ca4	4 / 9
b21	magnigate	70753deaae8ee1ca25e4704d1cb8cfc3764cc0be	18 / 21
2017-03-30-Terror-EK-traffic.pcap			
c15	Terror EK	c0bfba00d42ec74fdc56961e6eefe5234db7cd81	12 / 15
c36	Terror EK	e978915cbf33886911c029af056caf50e2d738e2	28 / 36
2017-01-02-pseudoDarkleech-Rig-V-sends-Cerber-ransomware.pcap			
h47	Rig-V	5948519a0e3e6ed77ffce5b44f88716687739652	38 / 47
2016-12-30-EITest-Rig-E-sends-Chthonicp-banking-Trojan.pcap			
i67	Rig-E	00ce7434b92f628025acad9d66ac0880a2d8d04e	36 / 67
2016-07-25-Magnitude-EK-sends-Cerber.pcap			
j77	Magnitude EK	840d169f6d9f46582b5c1e3889e2f26f398a611c	47 / 77

∗ パケット番号は，該当ファイルのリクエスト開始およびレスポンス開始パケットの番号

表 3 クローキングの解析結果
Table 3 List of analysis results of cloaking.

ファイル	項目 1	項目 2
a27	○	○
a55	○	×
b9	×	×
b21	×	×
c15	○	○
c36	○	○
h47	○	○
i67	○	○
j77	△	×

△：得られた解析ログだけでは不十分だったケース

かし，参照置換手法で対処できない方法で参照が行われており，`location` オブジェクトの参照捕捉に失敗した．

5.2.2.3 ケーススタディ：ファイル c15

本ファイルは，Terror EK によるリダイレクトページの HTML ファイルである．解析を試みたところ，フォーム送信（`HTMLFormElement.submit`）によるページ遷移が発生し，ブラウザのコンソールログに出力された解析ログが消失した．このため，コンソールログを保持する機能を有効にするか，または手動でページ遷移を抑制するための修正を施す必要があった．本評価では，`submit` を再定義するコードの追加によりページ遷移を抑制し，ログを得た．

5.2.2.4 ケーススタディ：ファイル j77

本ファイルは，5.1.2 項の後半の評価に用いたサンプル[*7]と同一で，Magnitude EK による攻撃プログラムを含む HTML ファイルである．解析の結果，参照置換手法により特定の API に対してフィンガープリンティングが行われたことを確認できた．しかし，5.1.2 項で述べたように，特定の API が未定義のため，図 10 のエラーによりプログラムが中断した．このエラー以降にほかにフィンガープリンティングが行われるか否かを確認するために，プログラムがエラーなく一通り実行されるように未定義の API を自分で定義したうえで，再度解析する必要があった．このため，図 3 では評価結果を “△” とした．また，このフィンガープリンティングがクローキングに用いられるか否かについては，解析ログから推測できなかった．

[*7] ファイル名：`2016-07-25-Magnitude-EK-more-html.txt`

5.2.3 考察と課題

クローキングの解析結果として，表 3 に示したとおり，9 ケース中 5 ケースは大きな問題なく解析できた．解析できた 5 ケースについては，解析ログを確認するだけでクローキングの内容をほとんど推測できており，提案システムによる解析は実用的であるといえる．

他の 4 ケースについては，解析手法をうまく適用できないか，得られた情報が不十分であり，クローキングの解析としては課題が残る結果となった．しかし，コード解析支援の観点では，手動による詳細な解析を支援する情報が得られているといえる．また，提案手法による解析は，API 操作に関する動的解析であるため，挙動とコードの紐づけが難しい．このため，挙動とコードを紐付けられるような情報を得られれば，コードの静的解析にも有用な解析支援システムになりうる．以下に，ケーススタディにより，明らかとなった提案システムにおける課題をまとめる．

(1) 参照置換手法の適用により，コード中でグローバルに

定義される関数や変数がグローバルでなくなり，コードの挙動が変わる問題への対処

(2) 参照置換手法で捕捉できない方法によるオブジェクト参照への対処

(3) リダイレクトなどページ遷移時に，コンソールログを保持する機能がないブラウザにおいて，ログが消失する問題への対処

6. おわりに

本論文では，解析妨害 JavaScript コード解析支援システムを提案し，既存の JavaScript コードの解析手法における課題へ対処した．提案システムにより実現した解析手法のコンセプトは，JavaScript コードによるブラウザ API 操作を網羅的に捕捉し，API 操作ログを出力することにより，JavaScript コードの挙動を把握できるようにすることである．このコンセプトによる解析を実現するために，Proxy 手法と参照置換手法を提案した．具体的には，API 操作の捕捉に Proxy オブジェクトを利用することで，既存手法では捕捉しきれない API 操作を捕捉する．また，変数の参照を置き換えることにより，置き換え禁止の API に対する API 操作を捕捉可能にする．

評価として，提案した 2 つの解析手法を実現した提案システムで JavaScript コードの解析を行い，提案システムの有用性を評価した．具体的には，難読化された JavaScript コードの解析，およびクローキングを行う JavaScript コードの解析を行い，解析結果について考察した．難読化解析においては，提案システムにより得られるブラウザ API 操作ログが，さらなる解析に有用な情報を含むことを示した．また，クローキング解析においては，過半数の場合で問題なく解析できることを確かめ，また，提案システムにおけるいくつかの課題を明らかにした．

残された課題として，5.2.3 項で述べた 3 つの対処の実現がある．

謝辞 本研究成果は，国立研究開発法人情報通信研究機構（NICT）の委託研究「Web 媒介型攻撃対策技術の実用化に向けた研究開発」により得られたものです．

参考文献

[1] Cisco: Cisco 2016 Annual Security Report (online), available from ⟨https://www.cisco.com/c/m/en_us/offers/sc04/2016-annual-security-report/⟩ (accessed 2017-08-08).

[2] Trustwave: 2017 Global Security Report (online), available from ⟨https://www2.trustwave.com/2017-Trustwave-Global-Security-Report.html⟩ (accessed 2017-07-07).

[3] 高田雄太，秋山満昭，針生剛男：ドライブバイダウンロード攻撃に使用される悪性な JavaScript の実態調査，電子情報通信学会技術研究報告，Vol.113, No.502, pp.59–64 (2014).

[4] Symantec: Symantec Global Internet Security Threat Report: Trends for 2008, Volume XIV, April 2009 (online), available from ⟨https://www.symantec.com/security-center/archived-publications⟩ (accessed 2017-07-07).

[5] 柴田龍平，羽田大樹，横山恵一：Js-Walker：JavaScript API hooking を用いた解析妨害 JavaScript コードのアナリスト向け解析フレームワーク，コンピュータセキュリティシンポジウム 2016 論文集，Vol.2016, No.2, pp.951–957 (2016).

[6] Malware-Traffic-Analysis.net (online), available from ⟨http://www.malware-traffic-analysis.net/⟩ (accessed 2017-08-08).

[7] 玉田春昭, 中村匡秀, 門田暁人, 松本健一：Java クラスファイル難読化ツール DonQuixote，ソフトウェア工学の基礎 XIII，日本ソフトウェア科学会 FOSE2006，pp.113–118 (2006).

[8] 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，情報処理学会研究報告，Vol.2014-DPS-161, No.17, pp.1–7 (2014).

[9] Takata, Y., Akiyama, M., Yagi, T., Yada, T. and Goto, S.: Website Forensic Investigation to Identify Evidence and Impact of Compromise, *Proc. 12th EAI International Conference on Security and Privacy in Communication Networks* (*SECURECOMM 2016*), pp.431–453 (2016).

[10] 西尾祐哉，廣友雅徳，福田洋治，毛利公美，白石善明：悪性 Web サイトを分析するためのマルチ環境解析における通信ログ解析の効率化，コンピュータセキュリティシンポジウム 2016 論文集，Vol.2016, No.2, pp.496–502 (2016).

[11] Xu, W., Zhang, F. and Zhu, S.: JStill: Mostly Static Detection of Obfuscated Malicious JavaScript Code, *Proc. 3rd ACM Conference on Data and Application Security and Privacy* (*CODASPY'13*), pp.117–128 (2013).

[12] Takata, Y., Akiyama, M., Yagi, T., Yada, T. and Goto, S.: Fine-Grained Analysis of Compromised Websites with Redirection Graphs and JavaScript Traces, *IEICE Trans. Information and Systems*, Vol.E100-D, No.8, pp.1714–1728 (2017).

[13] Low, D.: Protecting Java Code via Code Obfuscation, *Crossroads*, Vol.4, No.3, pp.21–23 (1998).

[14] Pellegrino, G., TschürtzEric, C. and Rossow, B.: jÄk: Using Dynamic Analysis to Crawl and Test Modern Web Applications, *Proc. 18th International Symposium on Research in Attacks, Intrusions and Defenses* (*RAID 2015*), pp.295–316 (2015).

[15] Blake Hartstein: jsunpack — A generic JavaScript unpacker (online), available from ⟨http://jsunpack.jeek.org/⟩ (accessed 2017-07-07).

[16] Relentless Coding: JSDetox (online), available from ⟨http://www.relentless-coding.org/projects/jsdetox/⟩ (accessed 2017-07-07).

[17] Ecma International: ECMAScript® 2015 Language Specification (online), available from ⟨http://www.ecma-international.org/ecma-262/6.0/⟩ (accessed 2017-1-20).

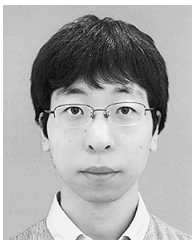
[18] 高田雄太，寺田真敏，松木隆宏，笠間貴弘，荒木粧子，畑田充弘：マルウェア対策のための研究用データセット ～MWS Datasets 2018～，情報処理学会研究報告，Vol.2018-CSEC-82, No.38, pp.1–8 (2018).

推薦文

難読化 JavaScript 解析における現状の課題を正確に把握しつつ，JavaScript と Web ブラウザの実装の理解にも

とづき，実ブラウザを利用して難読化 JavaScript の挙動を網羅的に解析可能なシステムを提案している．対象の JavaScript を提案システムで解析可能にするために数百行程度のコード片を挿入するだけで済むように設計されており，非常に実用性が高い．以上のことから，推薦論文に推薦いたします．

（コンピュータセキュリティシンポジウム 2017（CSS2017）
プログラム委員長　須賀 祐治）



上川 先之

2016 年岡山大学工学部情報系学科卒業．2018 年岡山大学大学院自然科学研究科博士前期課程修了．同年日本電信電話（株）入社，NTT セキュアプラットフォーム研究所にてサイバー攻撃対策技術の研究開発に従事．

秋山 満昭　（正会員）

2005 年立命館大学工学部卒．2007 年奈良先端科学技術大学院大学情報科学研究科修士課程了．同年日本電信電話（株）入社，NTT 情報流通プラットフォーム研究所にてマルウェア対策技術の研究開発に従事．2016 年 NTT セキュアプラットフォーム研究所特別研究員．2019 年同所上席特別研究員．主としてサイバー攻撃対策技術の研究開発に従事．博士（工学）．電子情報通信学会，IEEE 各会員．

山内 利宏　（正会員）

1998 年九州大学工学部情報工学科卒業．2000 年同大学大学院システム情報科学研究科修士課程修了．2002 年同大学院システム情報科学府博士後期課程修了．2001 年日本学術振興会特別研究員（DC2）．2002 年九州大学大学院システム情報科学研究院助手．2005 年岡山大学大学院自然科学研究科助教授．現在，同准教授．博士（工学）．オペレーティングシステム，コンピュータセキュリティに興味を持つ．2010 年度 JIP Outstanding Paper Award，2012 年度情報処理学会論文賞等受賞．電子情報通信学会，ACM，USENIX，IEEE 各会員．本会シニア会員．