# 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

柴田 龍平†1　羽田 大樹†1　横山 恵一†1

**概要**： 組織において Drive-by Download 攻撃によるマルウェア感染のインシデントが発生した場合，攻撃に使用された JavaScript を解析することがある．攻撃に使用されたスクリプトからリダイレクト先の URL を特定し，プロキシサーバー等のログと照合して影響範囲を特定するが，スクリプトには難読化が施されており，さらにブラウザの User-Agent, インストールされた Plugin などの環境情報を使用して特定環境でのみコードが実行される場合がある．

こうした解析妨害 JavaScript に対して従来手法では特定の条件でのみ発動する転送先 URL の抽出やその条件の特定を行う上で手動での解析が必要であった．

本研究では，JavaScript API hooking により実行環境を偽装して動的解析することで，難読化を解除したコードと特定の条件でのみ発動するリダイレクト先の URL を抽出する手法を提案する．評価を行った結果，複数の Exploit Kit 検体の解析において既存手法を上回る URL を自動で抽出できることを示すことができた．また，提案手法を用いて端末の変更を伴わずに解析環境を偽装し，JavaScript の実行結果を変化させられることを示した．

**キーワード**： JavaScript，Drive-by Download，インシデントレスポンス

# Js-Walker - Analyzing Anti-analysis JavaScript framework for analysts with JavaScript API hooking

Ryuhei Shibata†1　Hiroki Hada†1　Keiichi Yokoyama†1

***Abstract***: Anti-analysis JavaScript codes are often used in recent Exploit Kits that are used for drive-by download attack. Incident response team analyze these JavaScript codes and extract URLs when drive-by download incident occurs in the organization, then compare to surrounding trails for instance proxy logs, and finally judge the impact for the client. However, these JavaScript codes are well-obfuscated, and change behaviors regarding to user environment. In general, to extract property URLs or deobfuscated codes from these JavaScript code, it is required to understand conditions and to analysis in the same environment with infected client.

In this paper, we propose analysis technic with environment disguising using JavaScript API hooking, and easily extract URLs and. deobfuscated codes appears in specific environment.

The evaluation result shows proposed approach help analysts gain more URLs from anti-analysis JavaScript than existing approach, and proposed approach could change JavaScript code behavior by environment disguising technic.

***Keywords***: JavaScript, Drive-by Download, Incident Response

## 1. はじめに

### 1.1 Drive-by Download におけるインシデント対応

2015 年に Microsft が検出した脆弱性を用いた攻撃のうち，最も数を占めていたのが Exploit Kit による Drive-by Download である[1]．組織において Drive-by Download 攻撃によるマルウェア感染のインシデントが発生した場合，攻撃に使用された JavaScript を解析することがある．攻撃に使用されたスクリプトからリダイレクト先の URL を特定し，プロキシサーバー等のログと照合して影響範囲を特定するが，スクリプトには難読化が施されており，さらにブラウザの User-Agent，インストールされた Plugin などの環境情報を使用して特定環境でのみコードが実行される場合がある．このようなスクリプトの解析では，スクリプトを実行せずに解析者が解読する静的解析と，スクリプトを実行して挙動を観察する動的解析の 2 つのアプローチが存在する．静的解析では詳細な挙動が把握できるが解析に時間を要する．難読化が施されている場合は解析にかかる時間はさらに増大する．一方，動的解析では大まかな挙動が把握できるものの，対象のスクリプトに実行環境により挙動を変える機能が含まれている場合，解析環境での挙動が被疑端末での挙動と異なる場合がある．そのため，必要に応じて静的解析と動的解析を組み合わせて解析を行う．

### 1.2 解析妨害 JavaScript

2015 年に猛威を振るった Angler，RIG，Nuclear に代表される Exploit Kit においては，解析妨害を施した JavaScript が用いられるケースが多い[2]．また，改ざんペ

†1 NTT セキュリティ・ジャパン株式会社
NTT Security (Japan) KK

ージから Exploit Kit への転送に用いられる転送スクリプトにも解析妨害 JavaScript が埋め込まれるケースが確認されている[3]．こうした JavaScript では難読化，フィンガープリンティング，クローキングという手法でアナリストによる解析やセキュリティ製品による検査から自身のコードの保護を行っている[4]．

### (1) 難読化

JavaScript の文字列としてコードを含める，もしくは DOM の中にコードをエンコードして格納することで，実際に実行されるコードを秘匿する．こうしたコードは図 1 に示すように，DOM から要素を抽出して，Base64 や Unicode のエンコード方式で文字列に変換した後，JavaScript コンテキストで該当文字列を評価することで実行される．この評価は，eval や Function などを用いて直接コードを実行する方法と，appendChild などを用いて script タグを DOM 中に埋め込むことで間接的にコードを埋め込む方法の二通りが存在する． 近年の Exploit Kit においては，内部で実行された JavaScript コードがさらに難読化されており，これらの処理を複数回行うことで最終的なコードが実行されるケースもある．

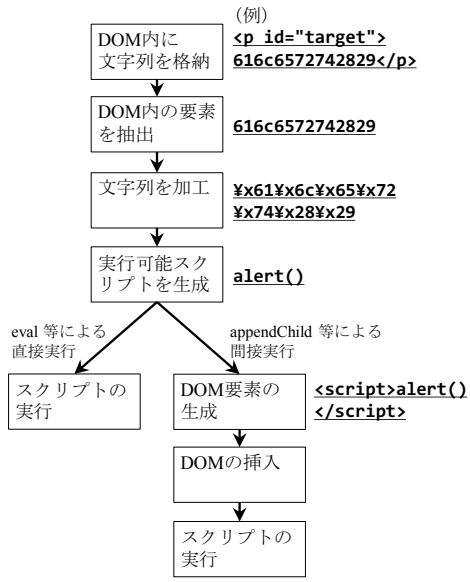


図 1　難読化 JavaScript コード実行の流れ

### (2) フィンガープリンティング

JavaScript コード中で User-Agent やインストールされた plugin 等のオブジェクトを参照することでユーザが利用しているブラウザの環境を特定し，これに応じたコードを実行することがある．Angler Exploit Kit に用いられた JavaScript の難読化を解除した後のコードを図 2 に示す．ここでは，navigator.userAgent を用いて User-Agent を参照し，Internet Explorer を検出することで条件分岐を行っている．

```
function xTrueV() {
    if (navigator.userAgent.indexOf('MSIE') == -1 && navigator.ap
pVersion.indexOf('Trident/') == -1) {
        return;
    }
```

図 2　Angler Exploit Kit の Internet Explorer 検出ロジック

### (3) クローキング

インシデント対応における Exploit Kit の調査では被疑 URL に対して後からアクセスすることがあるが，これに対抗するため，アクセス毎に返答コンテンツを変化させたり，コンテンツを一度しか返答しなかったりという処理が行われている．こうした動作はクローキングと呼ばれる．クローキングが施されたサイトを Web クローラ等で解析しても，うまく攻撃が再現できない場合があり，インシデントが発生した際に取得したパケットキャプチャ等を用いた解析が求められる．

## 1.3 本研究における目的

本研究では，JavaScript API hooking を利用して実行環境を偽装することで，解析妨害が施された JavaScript から難読化を解除した後のコードを抽出し，特定の条件でのみ発動する転送先 URL の抽出やその条件の特定を行う手法を提案する． インシデント対応で影響範囲を特定する Exploit Kit の解析者が利用することを想定し，市販の PC で支障なく解析できることを想定する．

# 2. 関連研究

## 2.1 JavaScript コードの解析に用いられるツール

こうした解析妨害 JavaScript の解析に利用される既存のツールとしては Jsunpack-n[5]，Revelo[6]といったツールが知られている．

Jsunpack-n は全自動でファイルに含まれる難読化解除後の JavaScript コードを出力するツールであり，PDF，HTML，SWF のような入力に対応している．Mozilla による JavaScript エンジンである SpiderMonkey[7]上でのコードの実行結果を元に Yara[8]を用いた検査を行う．Yara は対象ファイルやメモリ中から特定の文字列やバイト列をスキャンし，マッチング結果を出力するマルウェア解析用のツールであり，検査の結果内部で実行された JavaScript やシェルコードを自動で出力することができる．

Revelo はデバッガ相当の機能を持った JavaScript の解析ツールであり，コード実行によって埋め込まれる DOM の出力や特定の関数呼び出しの傍受などの特徴を持っている．ただし，多重難読化が施された JavaScript においては手動での解析が要求される．

インシデント対応においてアナリストはこれらのツールを用いて URL や難読化解除後のソースコードを抽出できる場合がある．ただし，解析妨害 JavaScript にフィンガープリンティングによる条件分岐が含まれている場合は，被害を受けた端末と同一の環境を再現するか，Internet Explorer や Google Chrome 等ブラウザの開発者ツール等を利用したデバッグで詳細な解析を行う必要がある．

### 2.2 先行研究

Pellegrino ら[9]は JavaScript の API hooking を用いて転送先 URL を取得する手法を提案している．JavaScript ではあらかじめ定義されている関数を再定義して処理を置き換えることができる．例えば図 3 のコードでは document.write 関数が独自の関数に置き換えられた結果，ドキュメントボディに hello という文字列が表示される前にアラートが表示される．これは動的言語としての ECMA Script の仕様であり，多くのブラウザで同様の置き換えが可能になっている[10]．API hooking を用いた手法の利点として JavaScript のエミュレーションのための大規模なコードを書く必要がなく，機能追加が非常に容易である点が挙げられる．

```
_write = document.write;
function f(arg){
    alert(arg);
    _write(arg);
}
document.write = f;
document.write(“hello”);
```

図 3　関数再定義による置き換え

Pellegrino らの手法ではユーザによる操作で発生する Ajax 通信の際の URL 取得のため，XMLHttpRequest オブジェクトの open/send メソッドおよび HTMLElement.prototype のイベントハンドラ登録関数のフックを利用している．

### 2.3 既存手法の課題

2.1 節で示したツールでは，難読化とフィンガープリンティングの両方が施された解析妨害 JavaScript に対して，実行環境が攻撃者の意図した条件に当てはまらないと目的のコードが実行されず，正しい転送先 URL を取得できないという課題があった．特定条件下でのみ転送が発生する URL や難読化解除後コードを抽出するためには，被疑端末と同様の環境を再現してコードを実行する必要があるが，インシデントの度にユーザにヒアリングを行い，同等の実行環境を用意して解析するのは困難である．想定される環境パターンの解析用マシンを事前に準備するという手法も考えられるが，多くの組織では想定される全てのクライアント環境を網羅的に再現することは現実的ではない．

また，2.2 節で示した Pellegrino らの手法については一般的な Web アプリケーションに対し，脆弱性スキャンを実施する際の URL 抽出を目的とした手法であり，一般的な Web アプリケーションで発生するリダイレクトに関しては有効に動作するものの，Exploit Kit における転送では DOM に動的に iframe タグや script タグ，object タグを埋め込むことで別のコンテンツを読み込む場合が多く，外部へのリダイレクトを抽出する場合は該当タグの埋め込みについても考慮する必要がある．また，環境情報による条件分岐やクローキングも想定していないため，Exploit Kit に適用しても適切に URL を抽出することができない．

## 3. 提案手法

### 3.1 提案手法の要件

本研究では，解析妨害 JavaScript を含むページの HTML を入力して API hooking を用いた動的解析を行う手法を提案する．実際のインシデント発生時に解析妨害 JavaScript を解析する場合，下記のような機能が求められる．

- 難読化の検出と解除
- フィンガープリンティングの検出と条件の特定
- Web アクセスを伴わない解析
- 簡易な実行環境の変更
- リダイレクト先 URL 一覧の抽出

### 3.2 提案手法

提案手法の全体図を図 4 に示す．

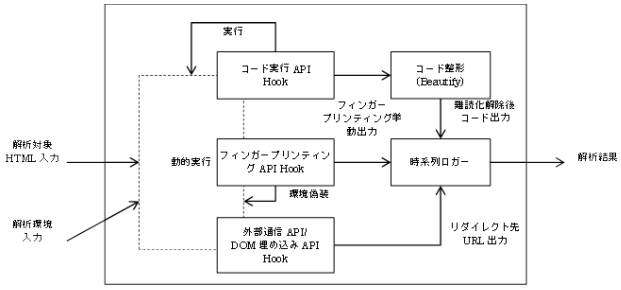


図 4　提案手法のシステム全体図

解析者は解析対象の HTML をブラウザ上で動作する GUI から入力する．また，必要に応じて解析時の User-Agent，appVersion などを入力する．入力を終えたら GUI 上のボタンを押下することで，提案システムが入力された HTML をパースし，HTML 中の JavaScript を実行する．このときあらかじめ，コード実行，フィンガープリ

ンティング，リダイレクトに用いられる API（関数）をフックしておき，フック関数ないでこれらの挙動を時系列ロガーに渡して非同期で GUI に出力することで難読化解除後のコード，フィンガープリンティング条件，リダイレクト先 URL を結果として得ることができる．

**(1) 難読化解除後のコード**

1.2 節(1)で示した直接，もしくは間接的に JavaScript コンテキスト評価するための関数（コード実行 API と呼ぶ）をフックすることで実行されるコードを抽出し，整形してログに出力する．抽出した JavaScript コードはさらに動的実行し，解析を継続する．詳細は 3.3 節にて示す．

**(2) フィンガープリンティング**

navigator.userAgent 等の環境情報が格納されているプロパティの参照（フィンガープリンティング API と呼ぶ）をフックし，解析対象 JavaScript コードによる環境情報を利用した条件分岐を補足する．加えて，API からの返答をあらかじめ設定していた値に変更することで解析対象の JavaScript コードの見せかけの実行環境を偽装する．詳細については 3.4 節にて示す．

**(3) リダイレクト先 URL**

Ajax に用いられる関数（外部通信 API），および DOM の埋め込みに用いられる関数（DOM 埋め込み API）をフックすることで別 URL への推移を抽出する．詳細は 3.5 節にて示す．

API フックを用いることで既存の JavaScript 実行エンジンが利用できるため，JavaScript のエミュレーションのための大規模なコードを書く必要がない．また，実行するためのブラウザ環境を柔軟に変更できるという利点がある．

### 3.3 難読化解除後コードの抽出

コード実行 API のフックを行うことで難読化解除後コードを抽出する．コード実行 API には eval 関数，Function などの直接実行型と script タグの埋め込みによる間接実行型が存在する．

以下は抽出を行った API の一覧である．

表 1　コード実行 API の一覧

| 種別 | 関数 |
|---|---|
| 直接実行 | eval |
| | setInterval |
| | setTimeout |
| 直接実行（関数定義のみ） | Function |
| | Function.prototype.constructor |
| 間接実行 | Element.prototype.appendChild |
| | Element.prototype.insertBefore |
| | Element.prototype.replaceChild |
| | document.write |
| | document.writeln |

上記のうち，直接実行型の API については引数の文字列を人間が読みやすいようにインデント付けなどの整形を行ったのち，時系列ロガーに渡す．間接実行型の API については，引数となる DOM 要素が script タグであるかどうか判別し，script タグ内にコードが含まれている場合のみ整形して時系列ロガーに渡す．いずれの API でも，ログ出力した後にコードを実行し，解析を継続する．

### 3.4 フィンガープリンティングの検出/環境偽装

Exploit Kit においてはブラウザの User-Agent やインストールされた plugin などの環境情報を調査するケースが多く，これらは navigator オブジェクトのプロパティによりアクセスできる．navigator オブジェクトのプロパティは getter/setter メソッドを通じてアクセスできるため，図 5 に示すように独自の getter/setter メソッドを持ったオブジェクトを Object.defineProperty 関数を用いて再定義することでフックを行う．ただし，当該手法は Safari では有効に機能しない．

本研究における実装では，以下のフィンガープリンティング API を抽出し，フックを行った．

- navigator.userAgent
- navigator.appName
- navigator.appVersion
- navigator.plugins
- document.referrer

```
var dummyUserAgent = “Mozilla/5.0 (Windows NT 6.1;
Trident/7.0; rv:11.0) like Gecko”;
Object.defineProperty(navigator, "userAgent", {
      set: undefined,
      get: function(){
          logger.log("UserAgent read attempt");
          return dummyUserAgent;
      }
});
```

図 5　フィンガープリンティング挙動の検出と環境偽装

### 3.5 リダイレクト先 URL の抽出

リダイレクト先 URL を抽出するために，外部通信 API と DOM 埋め込み API のフックを行う．外部通信 API として XMLHttpRequest.prototype の open 関数，send 関数をフックして JavaScript からの Ajax 通信を補足する．加えて，外部リソースを読み込む(1)object タグ，(2)iframe タグ，(3)applet タグ，(4)embed タグ，(5)script タグ，(6)img タグの埋め込みによって発生するリダイレクトを抽出する

ため，以下の DOM 埋め込み API をフックし，上記のタグの埋め込みがあった場合に読込先リソースの URL を抽出する．

- Element.prototype.appendChild
- Element.prototype.insertBefore
- Element.prototype.replaceChild
- *HTMLElemnt.prototype.innerHTML.set
- *HTMLElemnt.prototype.outerHTML.set
- document.write
- document.writeln

上記のうち appendChild，insertBefore，replaceChild，document.write，document.writeln についてはコード実行関数のうち間接実行を行う API と重複するため，script タグが埋め込まれた場合は src 属性を確認し，src 属性が存在する場合はリダイクト，src 属性が存在せず，タグ内に文字列が含まれている場合はコードの実行だと判別する．

DOM の埋め込みに使用される Element.prototype の innerHTML/outerHTML については navigator オブジェクトのプロパティと同様に getter/setter メソッドを通じてアクセスされるようになっている．ただし，これらの getter/setter については Internet Explorer や Google Chrome といった主要ブラウザで置き換えが制限されているため，BodyHTMLElement.prototype や DivHTMLElement.prototype などプロトタイプチェーンの上流のオブジェクトに対し，Object.defineProperty を用いてプロパティを定義することでフックを実現する．

上記の他に location オブジェクトを用いたリダイレクトが存在するが，プロパティの再定義によるフックができないため，onunload イベントを定義し，外部ページへの自動遷移を防止することで解析中に別ページにリダイレクトされることを防ぐ．なお，location オブジェクトを利用した遷移先を知りたい場合は onunload イベント後に意図的に遷移されることで URL が抽出できる．

## 4. 提案手法の評価

### 4.1 評価手法

本章では，公開情報[11]から取得した JavaScript が使用された Exploit Kit，および転送スクリプトに対して提案手法と Revelo を利用して URL を抽出して評価する．Exploit Kit および転送スクリプトの詳細については表 2 に示す通りである．

表 2　評価に用いた解析妨害 JavaScript の一覧

| 項番 | MD5 Hash | 種別 | 検知日 |
|---|---|---|---|
| 1 | 424f15690ee0ee71e1514f0d598dfabf | Nuclear Exploit Kit | 2015-11-15 |
| 2 | 0a6cf30ee3f242a86996b9239a1c70a4 | 転送スクリプト | 2015-11-15 |
| 3 | 615a62674200532d135db7bca333ee8d | Nuclear Exploit Kit | 2016-02-15 |
| 4 | 4254d83a5cbdbda2d0e5077caeafce2d | Rig Exploit Kit | 2016-02-23 |
| 5 | 3a87cdbff1e36fc59c220a5ac1119740 | Angler Exploit Kit | 2016-03-04 |
| 6 | 9579587cc732515603aa37e8edb2646c | 転送スクリプト | 2016-05-18 |
| 7 | 8c175748ee2db0ace075e6a088cce7af | Angler Exploit Kit | 2016-05-18 |
| 8 | e3a5f83a22d56dc152ef75984804692f | Rig Exploit Kit | 2016-06-26 |
| 9 | c0103370f0d85d2bb6ff1a1c1fc269fa | Magnitude Exploit Kit | 2016-08-03 |

比較手法については上記検体の HTML を入力に，複数の環境条件を指定した提案手法と Revelo を用いて解析を行う．また，提案手法の環境偽装機能の有効性を評価するため，提案手法では User-Agent や appVersion の値を変更して，InternetExplorer 7 環境と Internet Explorer 11 環境，Google Chrome での環境に偽装することで抽出 URL 数を比較する．それぞれの環境での各プロパティの設定値を表 3 に示す．

表 3　評価に用いる環境設定

| 環境 | UserAgent | AppName | AppVersion |
|---|---|---|---|
| IE7 | Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET 4.0C | Microsoft Internet Explorer | 4.0 (compatible; MSIE 7.0; Windows NT 6.1; Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET 4.0C |
| IE11 | Mozilla/5.0 (Windows NT 6.1; Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET 4.0C; rv:11.0) like Gecko | Netscape | 5.0 (Windows NT 6.1; Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET 4.0C; rv:11.0) like Gecko |
| Chrome | Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36(KHTML, like Gecko) Chrome/52.0.2743.116 Safari/537.36 | Netscape | 5.0 (Windows NT 6.1) AppleWebKit/537.36(KHTML, like Gecko) Chrome/52.0.2743.116 Safari/537.36 |

評価用の端末については OS X（2.7GHz Intel Core i5 / 16GB 1867MHz DDR3）の仮想マシン(1 Core / 2GB)を用いた．解析に用いたのアプリケーションのバージョンについては表 4 に示した通りである．

表 4　評価に用いた OS とアプリケーション

| 手法 | アプリケーション |
|---|---|
| 提案手法 IE7 | Internet Explorer 11.0.9600 |
| 提案手法 IE11 | Internet Explorer 11.0.9600 |
| 提案手法 Chrome | Google Chrome 52.0.2743 |
| Revelo | Revelo v0.6 |

### 4.2 結果

表 2 の検体の解析から各手法で得られた URL 数を表 5 に示す．網掛け部が該当の検体から最も多く URL を抽出できた手法である．表中では手動解析で得られた URL 数と比較し，URL を全て取得できたものを○，一部取得できたものを△，抽出できなかったものを×と表記する．

表 5　検体から抽出できた URL 数

| 項番 | 提案手法 IE 7 | 提案手法 IE 11 | 提案手法 Chrome | Revelo |
|---|---|---|---|---|
| 1 | ○(3/3) | ○(3/3) | ○(3/3) | ×(0/3) |
| 2 | ×(0/1) | ○(1/1) | ×(0/1) | ○(1/1) |
| 3 | ○(1/1) | ○(1/1) | ○(1/1) | ×(0/1) |
| 4 | ○(1/1) | ○(1/1) | ○(1/1) | ×(0/1) |
| 5 | ○(3/3) | ○(3/3) | △(2/3) | ×(0/3) |
| 6 | ×(0/1) | ○(1/1) | ×(0/1) | ○(1/1) |
| 7 | ○(1/1) | ○(1/1) | ○(1/1) | ×(0/1) |
| 8 | ○(1/1) | ○(1/1) | ○(1/1) | ×(0/1) |
| 9 | ×(0/1) | ×(0/1) | ×(0/1) | ○(1/1) |

項番 1，3，4，5，7，8 については提案手法が既存手法より多くの URL を取得できるという結果が得られた．項番 2, 6 については提案手法と既存手法で同数の URL が取得できた．項番 9 については既存手法でのみ URL を取得できる例であった．

### 4.3 考察

全体として提案手法による環境偽装がうまく働き，抽出できる URL 数に差異が生じているように見受けられる．

前述の項番 1，3，4，5，7，8 については 2 章で触れた script タグの埋め込みによるコード実行を利用した難読化が施されており，既存手法はそこで解析を終了しているように見受けられた．提案手法では script タグ埋め込みを用いた難読化についても問題なく解析を継続できたため，URL の抽出数に違いが生じたものと考えられる．

項番 2, 6 については Exploit Kit へのリダイレクタとして利用されている JavaScript コードであり，User-Agent 等の環境関数と cookie を参照した上で iframe を埋め込む挙動を持っていた．前述の script タグの埋め込みを用いた多段難読化が無かったため，既存手法でもうまく解析ができたものと推測される．

項番 9 については location オブジェクトを利用したリダイレクトであり，提案手法ではうまく URL が抽出できていない．既存手法はローカルホスト上のプロキシを経由して通信させることで location オブジェクト経由のリダイレクトを補足できているものと考えられる．一方で提案手法では onunload イベント発生時に警告を出すようにしているため，リダイレクトの発生には気付くことができるが，URL の網羅性という面で改善の余地がある．

とはいえ，複数の Exploit Kit 検体において既存手法を上回る数の URL を抽出できており，かつ環境の偽装を行う事で比較的簡単に検体の挙動を変化させることに成功していることから，提案手法は既存の手法を補完することができていると言える．また，評価項目としては提示していないが，フィンガープリンティング挙動の抽出と難読化解除後コードの抽出がされることにより，Exploit Kit 検体がどのような条件で挙動を変えるのかを推測することが容易になっている．

## 5. まとめ

本稿では近年の解析妨害が施された JavaScript コードに対し，JavaScript の API フックによる解析手法を提案した．改善の余地が存在するものの，複数の Exploit Kit 検体の解析において既存手法を上回る URL を自動で抽出できることを示すことができた．また，提案手法で解析環境を偽装し，実行結果を変化させられることを示した．

## 参考文献


[1] “Microsoft Security Intelligence Report”. https://www.microsoft.com/security/sir/archive/default.aspx,（参照 2016-08-11）

[2] “A closer look at the Angler exploit kit | Sophos Blog”. https://blogs.sophos.com/2015/07/21/a-closer-look-at-the-angler-exploit-kit/,（参照 2016-08-04).

[3] “ampaign Evolution: Darkleech to Pseudo-Darkleech and Beyond (PaloAlto Networks) “. http://researchcenter.paloaltonetworks.com/2016/03/unit42-campaign-evolution-darkleech-to-pseudo-darkleech-and-beyond/,（参照 2016-08-04).

[4] Yuta Takata，Mitsuaki Akiyama，Takeshi Yagi，Shigeki Goto: "Website Forensic Investigtion to Identify Evidence and Impact of Compromise" in Proceedings of the International Conference on Security and Privacy in Communication Networks (SecureComm)，2016

[5] “Jsunpack-n”．http://www.ipsj.or.jp/journal/submit/manual/j_manual.html,（参照 2016-08-04).

[6] ”Revelo” http://www.kahusecurity.com/2012/revelo-javascript-deobfuscator/,（参照 2016-08-04）

[7] “SpiderMonkey” https://developer.mozilla.org/ja/docs/SpiderMonkey, (参照 2016-08-12)
[8] “Yara” http://virustotal.github.io/yara/, (参照 2016-08-12)
[9] Giancarlo Pellegrino, Constantin Tschürtz, Eric Bodden, Christian Rossow: "jAEk: Using Dynamic Analysis to Crawl and Test Modern Web Applications" in The International Symposium on Research in Attacks, Intrusions and Defences(RAID), 2015
[10] Natalie Silvanovich(Google) : “Attacking ECMAScript Engines with Redefinition” in BlackHat US 2015
[11] “Malware Traffic Analysis”, http://www.malware-traffic-analysis.net/, (参照 2016-08-06)