修士論文

Drive-by-Download攻撃予測のための難読化JavaScriptの検知に関する研究

1410039　　本田 仁

主指導教員　面 和成
審査委員主査　面 和成
審査委員　宮地 充子
審査委員　平石 邦彦

北陸先端科学技術大学院大学

情報科学研究科

平成28年2月

概 要

近年，インターネットの普及に伴い，サイバー攻撃が蔓延している．その攻撃の一つが Drive-by-Download 攻撃 (DbD 攻撃) であり大きな脅威となっている．DbD 攻撃は，攻撃者によって悪性コードが仕組まれた web サイトにユーザがアクセスすると，web ブラウザやそのプラグインの脆弱性が悪用され，ユーザの無意識のうちに勝手にマルウェアがダウンロードされる攻撃である．DbD 攻撃では，攻撃プロセスにおいて JavaScript が悪用されることが多い．DbD 攻撃に用いられる悪意のある JavaScript は動作を不明瞭にし，検知を回避したり解析を妨害する目的で難読化が施されていることがある．

そこで，難読化された JavaScript と難読化が施されていない通常の JavaScript に出現する文字の差異に注目し，文字出現頻度を特徴量として，Support Vector Machine によって難読化 JavaScript の検知を試みる手法が提案されている．この既存手法には，特徴量計算が単純で計算コストが低い，JavaScript の構文解析やエミュレートが必要なく文法的に不完全な JavaScript も対象とできる，などの利点がある．また，この手法の評価においては，3 年分の D3M dataset を用いた交差検定による評価が行われている．しかしながら，この評価では，時系列を考慮せずに交差検定を行っており，新しいデータが学習データ，古いデータがテストデータになる場合が発生していると考えられる．これは，時系列的に道理に合わず，実世界に沿った評価ができていないと考えられる．

そこで，本稿では，この手法の評価方法である交差検定による実験・評価と，古いデータを学習データ，新しいデータをテストデータとするデータセットの時系列に基づく実世界の検知に準じた実験・評価を新たに行う．その結果を比較することで時系列を考慮した場合の影響を示し，手法評価の厳密化及び考察を行う．また，手法の拡張として，bigram による文字出現頻度を新たな特徴量として評価し，有効性を検討する．

目 次

第1章　はじめに

1.1　背景

近年，サイバー攻撃が蔓延し，その被害が増加している．その攻撃の一つが Drive-by-Download 攻撃 (DbD 攻撃) であり大きな脅威となっている．DbD 攻撃は，攻撃が仕組まれた web サイトにユーザがアクセスすると，web ブラウザやそのプラグインの脆弱性が悪用され，ユーザの無意識のうちに勝手にマルウェアがダウンロードされる攻撃である．ダウンロードされたマルウェアにより，個人情報・機密情報の漏えい，Bot ネットの一部になる等の被害が発生する．

IBM の報告によると，2014 年上半期では 21.9%，下半期では 11.3%の組織で，DbD 攻撃によるマルウェアダウンロードが発生している [5]．また，情報処理推進機構（IPA）が発表した情報セキュリティ 10 大脅威 2015[6] の一つである．このように，DbD 攻撃は猛威を振るっており，DbD 攻撃に対する防衛が必要となっている．

DbD 攻撃では，web ブラウザやそのプラグインの脆弱性を悪用することで，マルウェアダウンロードを行っているが，その際 JavaScript が用いられることがある．そして，悪用される JavaScript の多くは，動作を不明瞭にし，検知や解析を困難にするため難読化処理が施されている．従って，悪意のある難読化 JavaScript を検知することは，DbD 攻撃に対する防衛として重要となる．

1.2　目的

本研究が目指す最終的なゴールは，DbD 攻撃予測のための処理が単純かつ軽量な悪意のある難読化 JavaScript 検知手法の提案である．

DbD 攻撃予測のための既存研究には，大別し動的解析と静的解析の 2 種類が存在する．動的解析・静的解析にはそれぞれ，メリット・デメリットが存在するが，一般的に，処理が単純かつ軽量であるのは静的解析である．そこで，本研究では，静的解析に着目する．

難読化 JavaScript を検知する処理が単純かつ軽量である手法として，文字出現頻度をパラメータとして機械学習により検知を行う手法 [9] が提案されている．この手法では，難読化 JavaScript と難読化が施されていない通常の JavaScript との出現する文字に差異が生じることに注目し，JavaScript コード中の文字出現頻度を特徴量として，Support Vector Machine によって検知を試みている．この既存手法には，特徴量計算が単純で計算コスト

が低い，JavaScript の構文解析やエミュレートが必要なく文法的に不完全な JavaScript も対象とできる，などの利点がある．

この手法の評価においては，3 年分の D3M dataset を用いた交差検定による評価が行われている．しかしながら，この評価では，データセットの時系列を考慮せずに交差検定を行っており，新しいデータが学習データ，古いデータがテストデータになる場合が発生していると考えられる．これは，時系列的に道理に合わず，実世界に沿った評価ができていないと考えられる．

そこで，本稿では，既存研究 [9] における評価方法である交差検定による実験・評価と，古いデータを学習データ，新しいデータをテストデータとするデータセットの時系列に基づく実世界の検知に準じた実験・評価を行い，結果を比較することで時系列を考慮した場合の影響を示し，手法評価の厳密化及び考察を行う．また，手法の拡張として，bigram による文字出現頻度を新たな特徴量として評価し，有効性を検討する．

1.3 論文の構成

本論文の構成は以下の通りである．第 2 章では，準備として，本論文を読み進めるにあたって必要となる用語，概念，知識等を説明する．第 3 章では，DbD 攻撃検知に関するこれまでの関連研究について説明する．第 4 章では，本研究の対象とする既存手法 [9] とその問題点について詳述する．第 5 章では，手法評価の厳密化について述べる．第 6 章では，bigram を特徴とする評価について述べる．最後に第 7 章では，本研究の総括を述べる．

第2章　準備

2.1　Drive-by-Download攻撃

Drive-by-Download 攻撃 (DbD 攻撃) とは，攻撃が仕組まれた web サイトにユーザがアクセスするだけで，web ブラウザやそのプラグインの脆弱性が悪用され，ユーザの無意識のうちに勝手にマルウェアがダウンロードされる攻撃である．DbD 攻撃は主に以下に示す流れで行われ，最終的にマルウェアのダウンロードが発生する．DbD 攻撃の概要を図 2.1 に示す．

1. 攻撃者が正規 web サイトを改ざんし，悪性サイトへ誘導するコードを埋め込む．
2. ユーザが改ざんされたサイトへアクセスする．
3. 悪性サイトへ自動的にリダイレクトされる．
4. ブラウザ等の脆弱性を悪用され，ユーザの意図しないマルウェアダウンロードが発生．

DbD 攻撃では，web ブラウザやそのプラグインの脆弱性を悪用することで，マルウェアダウンロードを行っているが，その際 JavaScript が用いられることがある．

2.2　難読化JavaScript

DbD 攻撃に用いられる悪意のある JavaScript は動作を不明瞭にし，セキュリティツールの検知メカニズムを回避したり解析を妨害する目的で難読化が施されていることがある．難読化には，無意味なコードの追加や文字のエンコードなどを用い，人間が読むことを困難にした「不可読な難読化」と，一見難読化が施されていないコードに見えるものの，その挙動の理解が困難な「可読な難読化」が存在する．また，難読化技術自体は悪用目的だけでなく，知的財産保護の目的で用いられる場合も存在する．悪意のある難読化 JavaScript ではしばしば，難読化された文字列を JavaScript のビルトイン関数である unescape() 関数によるデコードや String.replace() 関数，String.charAt() 関数などの文字列操作により難読化を解除し，目的の動作を行うコードを生成した後，eval() 関数などによって実行が行われる．

図 2.2 に難読化 JavaScript の例を示す．この JavaScript では，unescape() 関数によってデコードした目的のコードを eval() 関数によって実行している．

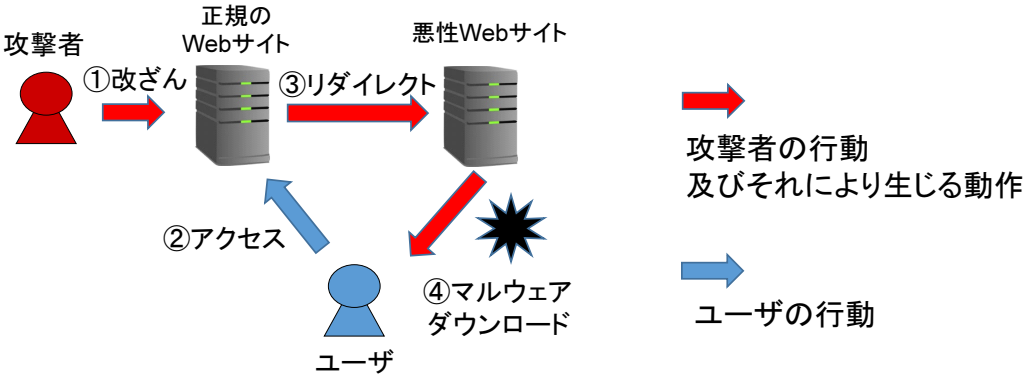

図 2.1: DbD 攻撃の流れ

```
eval(unescape('%65%76%61%6C%28%66%75%6E%63%74%69%6F%6E%28%61%55%
2C%75%76%62%2C%54%4F%68%2C%63%71%51%2C%45%68%43%2C%41%4D%29%7B%4
5%68%43%3D%53%74%72%69%6E%67%3B%69%66%28%21%27%27%2E%72%65%70%6C

                        ―中略―

%72%69%74%65%7C%77%69%64%74%68%7C%62%65%73%6C%6F%71%61%77%65%7C%
64%6F%63%75%6D%65%6E%74%7C%68%74%74%70%7C%73%72%63%7C%69%66%72%6
1%6D%65%7C%68%65%69%67%68%74%27%2E%73%70%6C%69%74%28%27%7C%27%29
%2C%30%2C%7B%7D%29%29'));
```

図 2.2: 難読化 JavaScript の例

2.3 D3M dataset

D3M dataset とは，マルウェア対策研究人材育成ワークショップ（MWS）によって提供されるマルウェア研究用のデータセット群である MWS datasets 内に含まれる DbD 攻撃に関する研究用データセットである．D3M dataset は，高対話ハニークライアントによって収集された DbD 攻撃に関するデータ群であり，以下に示す 3 つのデータから構成される．

1. 攻撃通信データ
   悪性 URL を巡回し得られた DbD 攻撃に関する pcap 形式の通信データ

2. マルウェア情報
   収集されたマルウェアのハッシュ値等の情報

3. マルウェア通信データ
   収集されたマルウェアを動作させた際の pcap 形式の通信データ

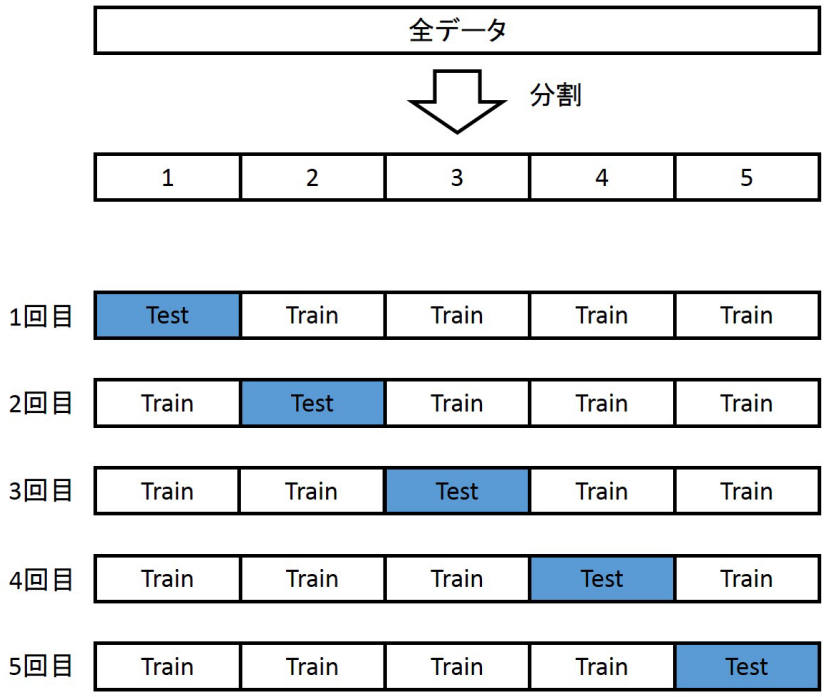

図 2.3: 5 分割交差検定

2.4 K 分割交差検定（K-Fold Cross Validation）

K 分割交差検定とは，ある識別モデルの未知のデータに対する性能を推定する際に用いられる検定法である．K 分割交差検定では，データの集合を K 個に分割し，その内の 1 つをテストデータとして用い，残りの K-1 個のデータを学習データとして評価を行う．この評価を分割した K 個のデータがそれぞれ 1 回ずつテストデータになるように K 回行い，得られた結果を平均することで最終的な識別モデルの評価結果を得る．図 2.3 に 5 分割交差検定 (K=5) の概要図を示す．

2.5 機械学習

機械学習は大きく教師あり学習，教師なし学習に分けられる．教師あり学習では，学習データとして特徴ベクトルと正解ラベルの入出力のペアが与えられ学習を行い，テストデータの特徴ベクトルが入力されたときに正しい出力を得ることを目的とする．教師なし学習では，学習データに正解の情報がなく，入力されたデータにおいて規則性などの構造を求めることを目的とする．本稿では，分類問題を取り扱うので教師あり学習を用いる．

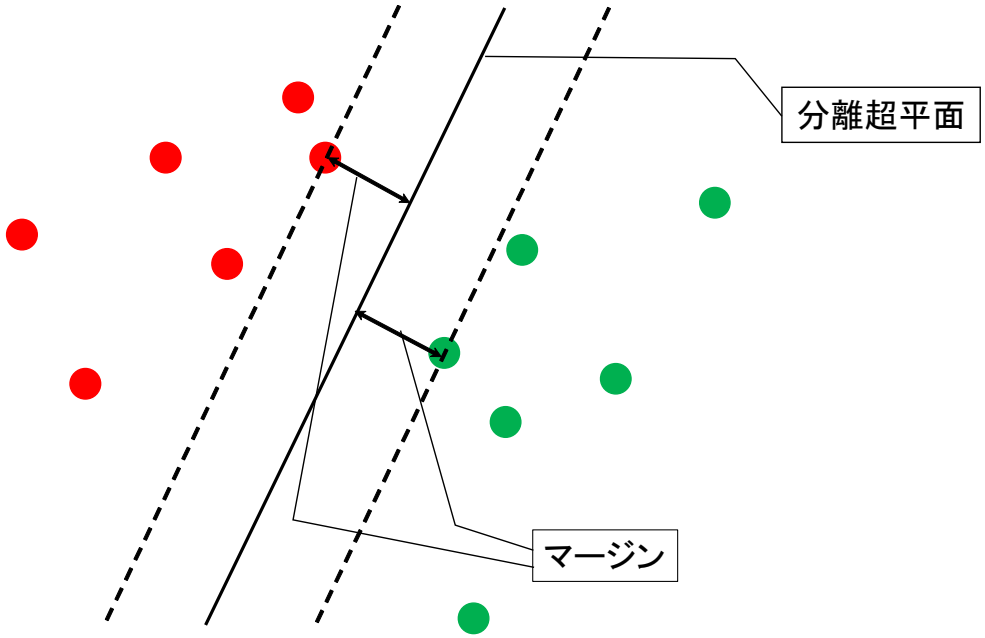

図 2.4: Support Vector Machine

本稿で用いる機械学習アルゴリズムについて以下に示す.

2.5.1 Support Vector Machine (SVM)

SVM は教師あり学習によるパターン識別手法の一つである (図 2.4). SVM では，入力された各学習データの内，分離超平面に最も近い距離にあるデータをサポートベクトルと呼び，サポートベクトルと分離超平面のマージンが最大となるように超平面を決定する. その分離超平面によって 2 値分類を行う.

2.5.2 Naive Bayes

Naive Bayes はベイズの定理を基にした分類器であり，ある入力に対し，尤度が最大となるクラスに分類する. Naive Bayes には，特徴量がガウス分布に従っていると仮定した Gaussian Naive Bayes，ベルヌーイ分布を仮定した Bernoulli Naive Bayes，多項分布を仮定した Multinomial Naive Bayes がある. 今回，入力に用いる特徴量である文字出現頻度は連続値となるため，本稿では，Gaussian Naive Bayes を用いる.

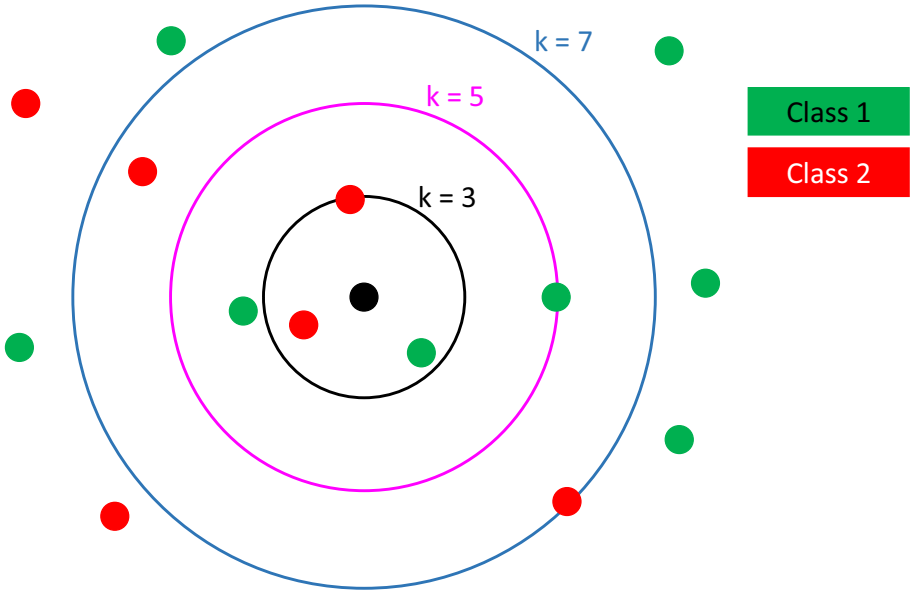

図 2.5: k-Nearest Neighbor

2.5.3 k-Nearest Neighbor

k-Nearest Neighbor では，分類対象のデータからの距離が短い k 個のデータを参照し，そのデータのうち最も多くのデータが属するクラスへ分類するアルゴリズムである．図 2.5 に例を示す．この例において，k=3 では，参照したデータのうち赤が多いため Class2 に分類され，k=5 では，緑が多いため Class1 に分類される．k=7 では，再び赤が多くなるため Class2 となる．2 値分類では，2 つのクラスで多数決を行うことになるので参照するデータが両クラスで同数とならないように，一般的に k は奇数をとる．

2.5.4 Decision Tree

学習データの値とそれが属するクラスの組から，データの分類をする木構造を生成するアルゴリズムである．エントロピーなどの分類基準を用い，分類効率が良い木となるように判別を行う中間ノードを決定していき，木を生成する．生成された木の葉ノードがクラスを表す．決定木のアルゴリズムには C4.5 や CART などがあり，分類の基準として C4.5 ではエントロピーを，CART ではジニ係数を用いている．

2.5.5 Random Forest

複数の Decision Tree を生成し，組み合わせることで 1 つの識別器とするアルゴリズムである．各々の木を生成する際に，入力された学習用データからランダムにデータをサン

プリングし，それぞれの木の生成を行う．生成された識別器を用いて識別を行う際には，各々の木で識別を行い，その結果を多数決することで，最終的な識別結果を出力する．

2.6 実装ツール

2.6.1 Python[10]

Python は，オープンソースのスクリプト言語である．変数の型やメモリ管理，ポインタなどをプログラマが明示的に指定しないため，これらをあまり意識せずソースコード記述することができる．また，構文の言語仕様としてインデントがブロックを表すようになっているため，自然とソースコードの体裁が良くなる．さらに，オープンソースとして公開されているため，だれでも容易に入手できる．このような特徴からプログラミング入門として用いられることが多い．機械学習や統計処理をはじめ，広範囲の分野に様々なライブラリが存在することも特徴の一つである．

本研究において，良性データセットの収集や，検知手法の評価実験などに用いるスクリプトはすべて Python で記述したものである．

2.6.2 scikit-learn[11]

scikit-learn は Python で利用可能なオープンソースの機械学習ライブラリである．SVM，Decision Tree，Random Forest，k-Nearest Neighbor，Naive Baise などの機械学習アルゴリズムを備え，分類，回帰，クラスタリングなどを行うことができる．また，正規化やスケーリング，2 値化などのデータの事前処理や，交差検定などのモジュールも用意されており，機械学習に関わる多くの処理を行うことができる．

本研究の機械学習においても scikit-learn を用いている．

2.6.3 HTMLParser[4]

HTMLParser は Python に標準で用意されたモジュールであり，HTML で記述されたファイルをパースする HTMLParser クラスが定義されている．HTMLParser クラスは，HTML データが入力されると，タグが開始したとき，開始タグと終了タグに挟まれたデータが見つかったとき，及びタグが終了したときに呼び出されるメソッドが定義された基底クラスである．基底クラスの実装では，これらのメソッドは所定のタイミングで呼び出されるのみで何も行わない．そのため，ユーザが行いたい動作を実装するには，この基底クラスを継承した派生クラスで各メソッドをオーバライドする必要がある．

本研究では，良性 JavaScript のデータを収集する際に，この HTMLParser を用いている．

2.7 検知における評価指標

機械学習による悪性難読化 JavaScript の検知における評価指標について説明する．まず，検体分類の正誤について以下の 4 通りのケースが考えられる．

- True Positive (TP)
  悪性検体を正しく判断できた場合（真陽性）
- True Negative (TN)
  良性検体を正しく判断できた場合（真陰性）
- False Positive (FP)
  良性検体を悪性として誤判断した場合（偽陽性）
- False Negative (FN)
  悪性検体を良性として誤判断した場合（偽陰性）

これら 4 通りのケースの発生数から算出される値を検知における評価指標として用いる．用いる評価指標を以下に示す．

- Accuracy
  Accuracy は，分類対象の全検体に対し，どれだけ正しく分類できたかを表す精度である．

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

- Precision
  Precision は，陽性と判断された検体に対する真陽性の割合である．この数値が低いと，陽性と判断した検体のうち，多くが実際には陰性 (良性) であるということを表す．

$$Precision = \frac{TP}{TP + FP} \tag{2.2}$$

- Recall
  Recall は，悪性と判断されるべき検体 (悪性検体) の内，悪性であると正しく判断した割合である．

$$Recall = \frac{TP}{TP + FN} \tag{2.3}$$

- False Negative Rate (FNR)
  FNR は，悪性と判断されるべき検体 (悪性検体) の内，良性と誤判断した割合であ

る．この値が高くなると，多くの悪性検体を良性として誤判断しており，悪性検体に対する検知漏れが多いことを表す.

$$FNR = \frac{FN}{TP + FN} \tag{2.4}$$

Recall の算出式 (2.3) から分かるように，$FNR = 1 - Recall$ である．つまり，Recall も FNR どちらも悪性検体に対する精度指標であり，本質的には同じものである．

- False Positive Rate (FPR)
  FPR は，良性と判断されるべき検体 (良性検体) の内，悪性と誤判断した割合である．この値が高くなると，多くの良性検体を悪性として誤判断しており，良性検体に対する検知漏れが多いことを表す.

$$FPR = \frac{FP}{TN + FP} \tag{2.5}$$

上記の数式 (2.1)，(2.2)，(2.3)，(2.4)，(2.5) 中の TP, TN, FP, FN は，それぞれのケースの発生数を表す．それぞれの算出式から分かる通り，Accuracy 及び Precision については数値が高いほど，FNR・FPR については数値が低いほど検知における性能が高いことを表す.

2.8 unigram, bigram について

unigram は，ある文字列において，任意の文字が 1 文字続いた文字列のことであり，unigram では，ある文字の出現について，前の文字とは関係なく独立であると考える．よって，出現頻度を算出するなどの処理の際は，1 文字を 1 つのトークンとして取扱い処理を行う．bigram は，ある文字列において，任意の文字が 2 文字続いた文字列のことであり，bigram では，ある文字の出現確率には，その前の文字との関連性があると考える．よって，連続した 2 文字のペアを 1 つのトークンとして取扱い処理を行う.

これらを一般的に表し，ある文字の出現確率に，その文字の $N-1$ 個前の文字まで関連性があると考える場合が N-gram である．N-gram は，任意の文字が N 文字連続した文字列であり，N-gram においては，連続した N 文字を 1 つのトークンとして処理を行う．対象とする文字の種類数を m とすると，N-gram におけるトークンの種類数は m^N となる.

第3章　関連研究

3.1　動的解析と静的解析について

DbD 攻撃検知に関する研究は，大きく分けて動的解析と静的解析の 2 種類が存在する．動的解析は実際に検体を動作させ解析を行い，静的解析は検体を動作させずに解析を行う．以下に動的解析と静的解析の利点・欠点を示す．

1. 動的解析
    - 利点
        - 検体を動作させるため，難読化の影響を受けない．
        - 実際の挙動から特徴を取得できる．
    - 欠点
        - 静的解析に比べ，計算リソースを多く消費する．
        - 検体を動作させた際，悪影響が及ばないようにサンドボックス環境が必要．
        - 環境依存で動作する検体 (攻撃目標のプラグインが存在する場合のみ動作する検体など) の解析に弱い．
2. 静的解析
    - 利点
        - 動的解析に比べ，計算リソースを消費しない．
        - 検体を動作させないため，サンドボックス環境が不要．
        - 環境依存の検体の解析に強い．
    - 欠点
        - 難読化の影響を受けやすい（特に単純なパターンマッチングの場合）．
        - 実際の挙動から特徴を取得できない．

3.2　動的解析を用いた関連研究

3.2.1　XZZ'13[12]

[12] では，難読化 JavaScript コードは実行するうえで難読化解除される必要があり，解除プロセスでは特定の関数を呼び出す必要があることに基づき，関数呼び出しから特徴を

動的解析にて取得し，検知を試みている．関数呼び出しに関して，以下の 3 つの観点から特徴を抽出している．

1. 難読化された悪意のある引数
   unescape() 関数などのコードの難読化解除に用いられる関数には，難読化された悪性コードが渡される．そこで，関数に渡された引数の中に難読化されたものが存在するかをチェックする．

2. 難読化された関数の定義
   悪性コードの多くのケースでは，悪意のある関数は，その定義が難読化され，隠されていることが多い．この場合，JavaScript の実行時に悪意のある関数の定義がされるが，静的な観点からは未定義のように見える．そこで，関数定義の難読化による隠ぺいが存在するかをチェックする．

3. ビルトイン関数呼び出しの偽装
   難読化された悪性 JavaScript においては，静的解析を回避するため unescape() 関数や，eval 関数 () などの難読化解除に用いられるビルトイン関数の呼び出しが偽装されていることがある．この場合，実際には，ビルトイン関数を呼び出しているが，静的な観点からは呼び出してないように見える．そこで，ビルトイン関数の呼び出しの偽装が存在するかチェックする．

上記 3 つの特徴の解析の結果，いずれかの特徴のうち，どれか 1 つでも悪性であると判断した場合，悪性として分類する．

3.2.2 JCB'14[7]

[7] では，JavaScript インタプリタにより検査対象 JavaScript のオペコードを抽出し，そのオペコードの N-gram を特徴ベクトルとして，Support Vector Machine を用い，検知を試みている．以下に処理の流れ（図 3.1）を示す．

1. データ抽出
   web ページレンダリングプロセスにおいて，JavaScript インタプリタによって生成される JavaScript のオペコードを取得する．

2. データ削減
   効率化のためオペランド等を削除し，オペコードキーワードの配列を構築する．

3. 特徴抽出
   構築したオペコードキーワードの配列から N-gram によって特徴を抽出する．この際，N を増やすと，特徴空間全体のサイズが劇的に増加するので，N の最適値は，許容される効果と効率のトレードオフで経験的に決定する．

4. 特徴表現
   抽出した N-gram による特徴を，分類クラスのラベルを付与した特徴ベクトルとして表現する．σ を訓練データセット内の異なるオペコードの数とすると特徴ベクトル空間の次元数は σ^N となる．

5. 分類
   特徴ベクトルを SVM によって分類を行う．

3.3 静的解析を用いた関連研究

3.3.1 LJJ'09[8]

[8] では，難読化 JavaScript と通常の JavaScript における特定の JavaScript キーワードの出現頻度，文字数や空白の割合の違いに着目している．eval など 50 種のキーワードの出現頻度と表 3.1 に示す 15 種の特徴の計 65 種の特徴を算出し，機械学習により検知を試みている．

3.3.2 CCVK'11[3]

[3] では，動的解析のリソースの節約を目的とするフロントエンドフィルタとして，処理が高速な静的解析による検知手法を提案している．動的解析のフロントエンドフィルタとして機能させるため，悪性検体の検知漏れがないように FNR に重きを置いている．[3] では以下の 3 つのカテゴリから静的解析によって特徴を抽出し，機械学習によって DbD 攻撃の検知を試みている．

1. HTML ファイルから抽出した特徴
   iframe タグの数，小さい面積の HTML 要素の数，ページ内の空白の割合，HTML ファイルに含まれる URL の数など計 19 種類

2. JavaScript から抽出した特徴
   eval() 関数の出現数，長い文字列の数，setTimeout() 関数と setInterval() 関数の出現数，スクリプトの文字列の最大長など計 25 種類

3. URL およびホストベースから抽出した特徴
   URL の登録日，URL に現れる IP アドレス，DNS の A レコードの生存時間，DNS の NS レコードの生存時間など計 33 種類

上記の 3 つのカテゴリごとに機械学習の検知モデルを生成し，それぞれのカテゴリで最良の結果を出したモデルを採用している．3 つのカテゴリの検知モデルごとに悪性・良性の判断を行い，1 つでも悪性と判断したら悪性として分類する．これにより FPR は増えるが FNR が減る．

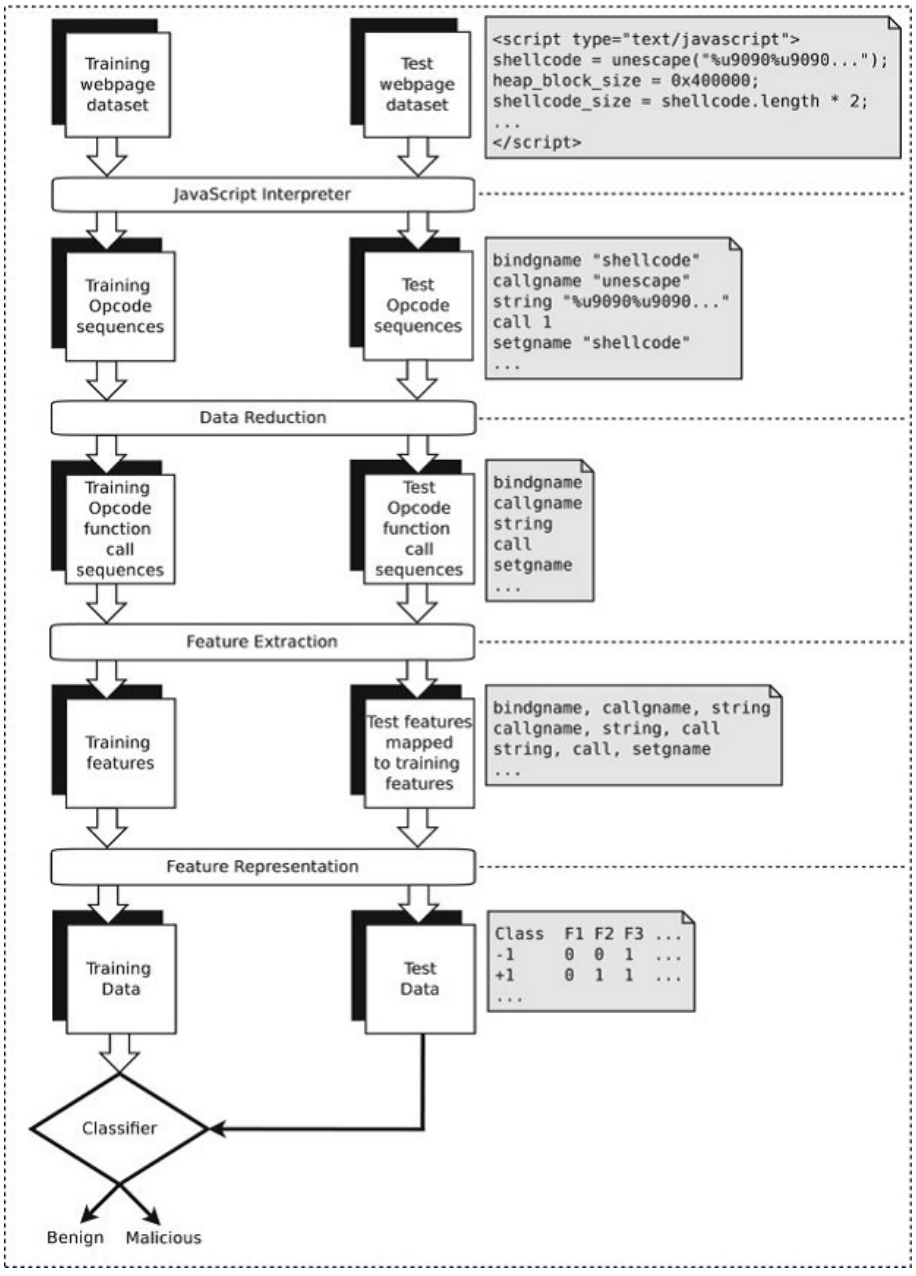
Training webpage dataset
Test webpage dataset
<script type="text/javascript">
shellcode = unescape("%u9090%u9090...");
heap_block_size = 0x400000;
shellcode_size = shellcode.length * 2;
...
</script>
JavaScript Interpreter
Training Opcode sequences
Test Opcode sequences
bindgname "shellcode"
callgname "unescape"
string "%u9090%u9090..."
call 1
setgname "shellcode"
...
Data Reduction
Training Opcode function call sequences
Test Opcode function call sequences
bindgname
callgname
string
call
setgname
...
Feature Extraction
Training features
Test features mapped to training features
bindgname, callgname, string
callgname, string, call
string, call, setgname
...
Feature Representation
Training Data
Test Data
Class F1 F2 F3 ...
-1 0 0 1 ...
+1 0 1 1 ...
...
Classifier
Benign
Malicious

図 3.1: [7] の処理の流れ

表 3.1: [8] において用いられる特徴

Feature	Description
Length in characters	The length of the script in characters.
Avg. characters per line	The average number of characters on each line.
Num. of lines	The number of newline characters in the script.
Num. of strings	The number of strings in the script.
Num. of unicode symbols	The number of unicode characters in the script.
hex or octal numbers	A count of the numbers represented in hex or octal.
% human readable	We judge a word to be readable if it is $> 70\%$ alphabetical, has $20\% <$ vowels $< 60\%$, is less than 15 characters long, and does not contain > 2 repetitions of the same character in a row.
% whitespace	The percentage of the script that is whitespace.
Num. of methods called	The number of methods invoked by the script.
Avg. string length	The average number of characters per string in the script.
Avg. argument length	The average length of the arguments to a method, in characters.
Num. of comments	The number of comments in the script.
Avg. comments per line	The number of comments over the total number of lines in the script.
Num. of words	The number of "words" in the script where words are delineated by whitespace and JavaScript symbols (for example, arithmetic operators).
% word not in comments	The percentage of words in the script that are not commented out.

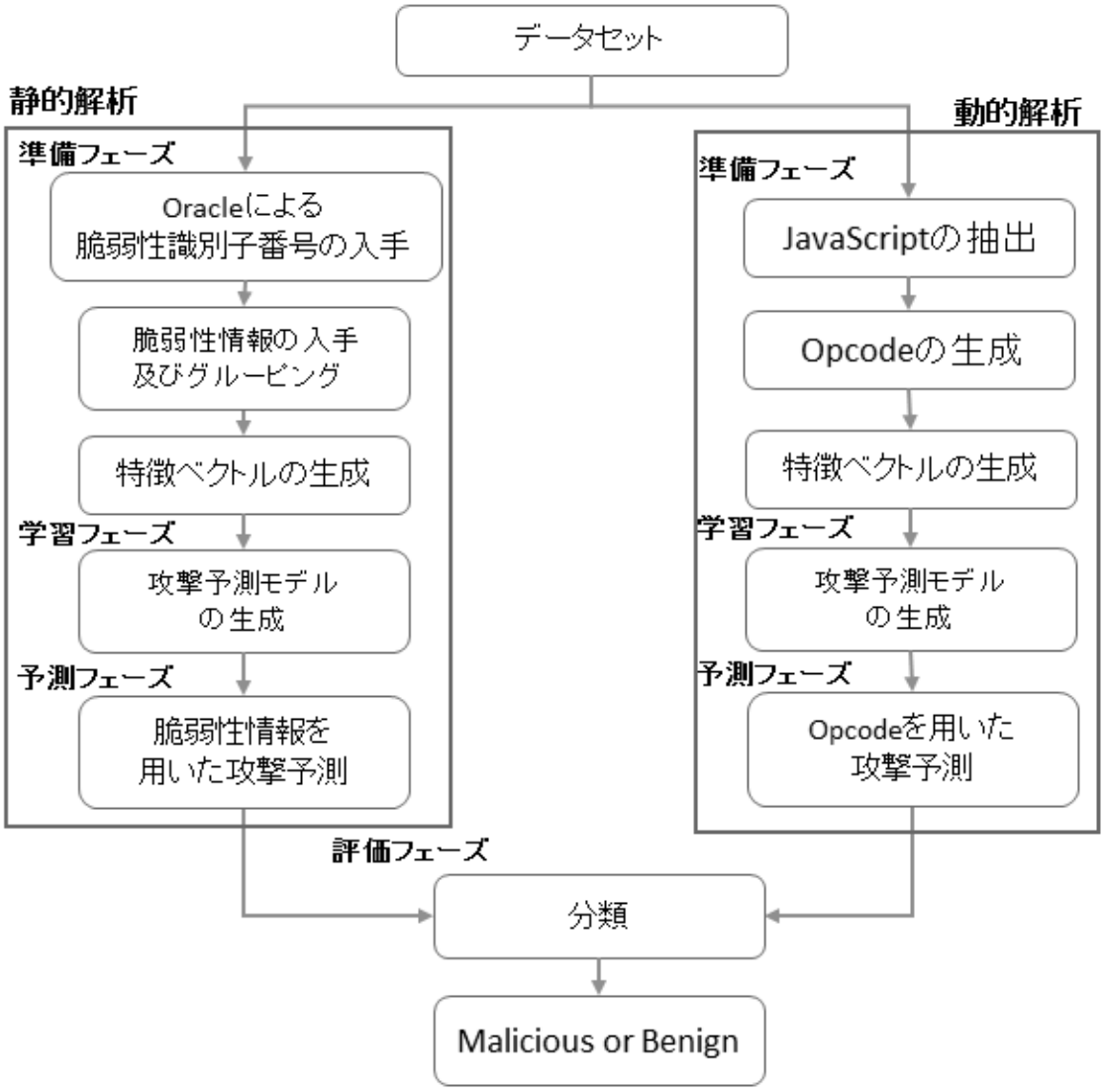
データセット
静的解析
準備フェーズ
Oracleによる
脆弱性識別子番号の入手
脆弱性情報の入手
及びグルーピング
特徴ベクトルの生成
学習フェーズ
攻撃予測モデル
の生成
予測フェーズ
脆弱性情報を
用いた攻撃予測
動的解析
準備フェーズ
JavaScriptの抽出
Opcodeの生成
特徴ベクトルの生成
学習フェーズ
攻撃予測モデル
の生成
予測フェーズ
Opcodeを用いた
攻撃予測
評価フェーズ
分類
Malicious or Benign

図 3.2: [1] の処理の流れ

3.3.3 NHKEIN'14[9]

[9] では，JavaScript 内の文字出現頻度を特徴ベクトルとして機械学習によって検知を試みている．本研究が評価の対象としている研究であるため，詳細は 4 章に示す.

3.4 動的解析と静的解析の組み合わせを用いた関連研究

3.4.1 AO'15[1]

動的解析と静的解析を組み合わせた手法として，[1] があり，この手法では，静的解析によって検査対象の URL から脆弱性情報を入手し特徴ベクトルを生成する，また，動的解析手法 [7] によって JavaScript のオペコードによる N-gram の特徴ベクトルを生成する．静的解析と動的解析によって得られた 2 つの特徴ベクトルを機械学習に用い，静的解析・動的解析のどちらかの特徴ベクトルにおいて陽性とされれば，DbD 攻撃であると判断している．図 3.2 に処理の流れを示す.

3.5 関連研究のまとめと本研究との関連

関連研究の比較を表 3.2 に示す.

動的解析と静的解析には，それぞれ利点と欠点が存在するが，本研究は処理が単純かつ軽量な悪意のある難読化 JavaScript 検知手法の提案を最終的なゴールとしているため，リソースの消費が少ない静的解析に着目する．[9] は，検知に用いる特徴が JavaScript コードにおける文字出現頻度のみであり，特徴量計算の処理が単純であり計算コストが低い．よって，先述した静的解析を用いた関連研究のうち，本研究が目指す検知手法に最も近いと言える．しかし，[9] の評価実験においては，3 年分のデータを用いているが，その時系列を考慮していないという問題点が存在する．これは時系列的に道理に合わず，実世界に沿った評価ができていないと考えられる．そこで本稿では，この手法に対してデータセットの時系列に基づく評価の厳密化を行う.

表 3.2: 関連研究の比較

	動的/静的	検知に用いる特徴
	問題点・欠点	
LJJ'09[8]	静的	JavaScript 中の特定のキーワードの出現頻度等
	キーワードの出現頻度は難読化の影響を受けやすい.	
CCVK'11[3]	静的	HTML ファイル，JavaScript，URL の 3 カテゴリから抽出した特徴
	多くの特徴を用いるため比較的特徴抽出のオーバヘッドが大きい.	
NHKEIN'14[9]	静的	JavaScript のコードにおける文字出現頻度
	評価実験においてデータセットの時系列を考慮していない.	
XZZ'13[12]	動的	難読化コードから抽出した関数呼び出しに関する特徴
	構文解析や，関数のフックなどの解析環境の準備が必要.	
JCB'14[7]	動的	JavaScript オペコードの N-gram
	解析の効率が，検体 JavaScript の実行時の効率に拘束される.	
AO'15[1]	動的＋静的	静的解析により抽出した脆弱性情報, 動的解析により抽出した JavaScript のオペコード
	脆弱性情報を特定する時間がボトルネックとなる.	

第4章　既存手法[9]とその問題点

4.1　既存手法[9]について

2.2 で述べたような難読化 JavaScript の特徴から，難読化された JavaScript と通常の JavaScript との文字出現頻度には差異があると推測される．そのため既存手法 [9] では，JavaScript の文字出現頻度を機械学習の入力特徴ベクトルとして採用している．

既存手法 [9] では，検体の JavaScript コードの文字を対象とし，空白と制御文字を除く 94 種類の ASCII 文字（ASCII 文字コードが 0x21〜0x7e の文字）について出現頻度を算出し特徴ベクトルとしている．この際，対象文字 i の JavaScript コード内における出現数を m_i とすると，JavaScript 内の対象文字の数 N と対象文字出現頻度 $F(i)$ は以下のように定義される．

$$N = \sum_i m_i \tag{4.1}$$

$$F(i) = \frac{m_i}{N} \tag{4.2}$$

$F(i)$ は $0 \leq F(i) \leq 1$ の値をとり，この $F(i)$ を Support Vector Machine（SVM）の入力特徴ベクトル（94 次元）として検知モデルを構築している．このように，1 文字を 1 つのトークンとして出現頻度を求めているため，unigram に相当する．

また，既存手法 [9] では，対象とする JavaScript の文字コードを UTF-8 と想定しており，マルチバイト文字は無視している．

4.2　既存手法[9]のメリット・デメリット

既存研究 [9] においては，手法のメリットとデメリットとして，以下のものが挙げられている．

- メリット
  - 特徴ベクトルの計算が単純であり高速
    検査対象の JavaScript の文字頻度のみを特徴とするため，テキストベースの

処理であり特徴ベクトルの計算コストが小さい．

– 文法的に不完全な JavaScript も検知対象にできる
　動的解析の場合，検査対象の JavaScript は，動的実行環境である JavaScript インタプリタが解釈できるよう誤りのないものである必要がある．しかし，[9] における文字出現頻度の特徴ベクトル算出は，静的解析であり，JavaScript の構文解析やエミュレートが不要であるため，文法的に不完全な JavaScript や途中で切れてしまっているような JavaScript も検知対象にすることが可能である．

- デメリット

– 難読化されていない JavaScript の検知が困難
　既存手法 [9] では，JavaScript の難読化によって文字出現頻度にその JavaScript 固有の特徴がみられると仮定してるため，難読化が施されていない JavaScript の場合は検知が困難である．

– 検査対象の JavaScript のファイルサイズによって影響を受ける
　検査対象の JavaScript のファイルサイズが小さい場合，その JavaScript を特徴づけるほどの文字数がなく，文字出現頻度による特徴抽出が困難である．

– 難読化解除のためのデータが Document Object Model (DOM) によって HTML エレメントから取得されている場合には検知が困難
　難読化の一手法に，難読化解除のためのデータが HTML エレメントの中にテキストとして埋め込まれているものがあり，DOM によってこれらのデータを取得し，難読化解除に用いる難読化手法がある．このような難読化手法では JavaScript には変換アルゴリズムしか含まれておらず，文字出現頻度による特徴が JavaScript にあらわれないと考えられるため，検知が困難になる．

4.3 検証実験

4.3.1 検証用 JavaScript データセット

[9] では，手法の検証実験用として，以下に示す手順で一般サイトの JavaScript と悪性難読化 JavaScript を収集しデータセットとしている．

- 一般サイトの JavaScript
　一般サイトの JavaScript として，Alexa[2] が公開しているアクセストップドメインのランキング上位 500 位のサイトのトップページにアクセスし，トップページ内の <script> タグに埋め込まれている JavaScript を収集している．また，<script> タ

表 4.1: [9] の実験用 JavaScript データセット

	良性	悪性
収集日	2013/11/18	2011/2/8-2013/2/26
スクリプト数	2786	330

グに src 属性による URL の記述が存在する場合，その URL にアクセスし JavaScript を収集している．

また，4.2 のデメリットに示したとおり，ファイルサイズが小さい場合，その JavaScript を特徴づけるほどの文字数がなく，文字出現頻度による特徴抽出が困難であると考えられるため，1KB 未満の JavaScript は除外している．

[9] では，これらの一般サイトの JavaScript を良性 JavaScript データサンプルとして取り扱っている．

- 悪性難読化 JavaScript

  D3M dataset の攻撃通信データのうち，2011 年から 2013 年までの 3 年分のデータを検証用データセットの対象としている．[9] では，これらの通信データに含まれる JavaScript のうち，jQuery 等の一般的なライブラリと思われるものを除外し，明らかに難読化されているとわかるもののみを悪性 JavaScript データサンプルとして採用している．また，良性 JavaScript データサンプルと同様に 1KB 未満のスクリプトは除外している．

  なお，[9] では，これらの悪性データサンプルとした JavaScript が，実際に悪意のある挙動を行うかどうかの検証は行われていない．

上記の良性データと悪性データを合わせて検証用 JavaScript データセットとしている．良性・悪性データの収集日とスクリプト数を表 4.1 に示す．

4.3.2 SVM による学習

まず教師データの作成を行い，その教師データを用い SVM による学習および交差検定を行っている．[9] では，SVM を扱うためのライブラリである libsvm を用い，SVM による学習を行っている．

1. 教師データの作成

   式（4.2）によって文字出現頻度 $F(i)$ を，その対象となっている 94 種の文字に対し算出し，SVM の入力ベクトルとする．よって入力ベクトルは 94 次元となる．検証用 JavaScript データセットの良性・悪性 JavaScript データそれぞれに対し入力ベクトルを求め，良性 JavaScript データに対しては $+1$ のラベルを，悪性 JavaScript データに対しては -1 のラベルを付与し教師データを作成している．

表 4.2: [9] における実験結果

	Result
Accuracy	98.84%
Precision	97.72%
Recall	94.35%

2. SVM のカーネルとパラメータの最適化

学習に用いる SVM のカーネルは，RBF（ガウス）カーネルを用いている．このとき，SVM のパラメータである C と γ を設定する必要があるが，[9] では，libsvm に付属している grid.py を用いることでグリッドサーチにより網羅的にこの 2 つのパラメータの探索を行っている．この際の交差検定の分割数は 5 としている．

4.3.3 実験結果

[9] では，作成された教師データを用いてグリッドサーチを行った結果，$C = 25.22$, $\gamma = 55.72$ のとき最も高い Accuracy が得られた．このときの結果を表 4.2 に示す．

4.4 問題点

既存研究 [9] においては，悪性データとしてマルウェア対策研究人材育成ワークショップ（MWS）によって提供されるマルウェア研究用のデータセットである MWS datasets 内の D3M dataset の 3 年分のデータを用い，交差検定によって手法の評価が行われている．

しかし，データセットの時系列を考慮せずデータを 1 つのまとまりにし，交差検定を行った場合，新しいデータが学習データとなり，古いデータがテストデータになる場合が発生し，時系列的に道理に合わず，実世界に沿った評価ができないと考えられる．

また，SVM のパラメータのチューニングにおいても，データセットの時系列を考慮せずデータを 1 つのまとまりにしたデータに対し，交差検定によって最適なパラメータを求めると，そのパラメータにはテストデータによるチューニングが内包されていることになる．実世界の検知においては，テストデータ，つまり検査対象のデータは悪性であるか良性であるか分からないため，学習用の教師データのみを用いてチューニングしなければならない．よって，SVM のパラメータのチューニングの点においても実世界に沿った評価ができていないと考えられる．

この 2 点から [9] の評価実験は，現実の攻撃検知にそぐわないと考えられるため，本研究ではデータセットの時系列を考慮した評価実験を行う．

第5章　手法評価の厳密化

5.1　手法評価の厳密化の目的

4.4 で述べたように，既存研究 [9] においては，交差検定によって評価が行われており，データセットの時系列が考慮されていない．そこで，本研究では，既存研究 [9] における評価方法である交差検定による実験・評価と，図 5.1 に示すように，古いデータを学習データ，新しいデータをテストデータとするデータセットの時系列に基づく実世界の検知に準じた実験・評価を行い，結果を比較することでデータセットの時系列を考慮した場合の影響を示し，手法評価の厳密化及び考察を行う．

5.2　評価実験に用いる JavaScript データセットについて

評価実験に用いる検証用 JavaScript データセットは，既存研究 [9] に則り，データの収集・構築を行った．検証用 JavaScript データセットは以下に示す良性 JavaScript データおよび悪性 JavaScript データから構成される．

5.2.1　良性 JavaScript

Alexa Top500[2] が公開しているアクセストップドメインリストの web サイト URL にアクセスし，トップページ内の <script> タグで埋め込まれた JavaScript，および <script> の src 属性に記された URL の JavaScript を収集したものから構成される．JavaScript の収集においては Python スクリプトを作成し，JavaScript の収集を自動化した．対象の URL から JavaScript データを収集する Python スクリプトの流れを図 5.2 に示す．既存研究 [9] に則り，スクリプトのデータサイズが小さい場合，特徴量がうまく抽出されないと考えられるため，データサイズが 1KB 未満のスクリプトは除外した．

5.2.2　悪性 JavaScript

2011 年-2014 年の D3M dataset の攻撃通信データから，悪性コードを含むと思われるものを抽出・収集し，悪性 JavaScript のデータセットとした．悪性 JavaScript のデータセットについても良性の場合と同様に，データサイズが 1KB 未満のスクリプトは除外した．

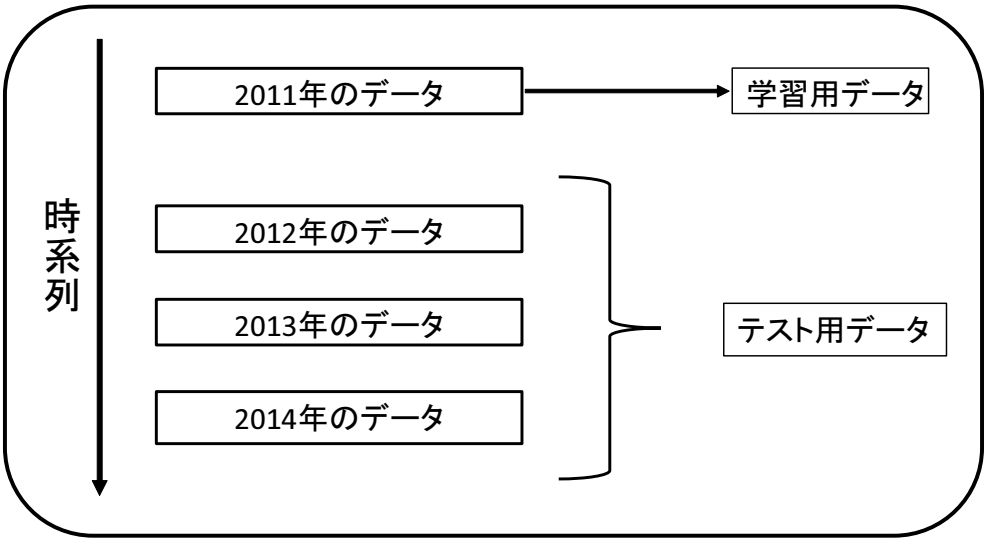

図 5.1: 時系列を考慮した評価

表 5.1: 実験用 JavaScript データセット

	良性	悪性
収集日	2015/6/9	2011/2/14-2014/4/11
スクリプト数	3344	906

5.2.3 検証実験用 JavaScript データセット

収集した良性 JavaScript と悪性 JavaScript を合わせて検証実験用 JavaScript データセットとした．良性・悪性 JavaScript の収集日とスクリプト数を表 5.1 に示す.

5.3 評価実験

既存手法 [9] のアルゴリズムを実装したスクリプトをフルスクラッチで作成し，データセットの時系列を考慮した場合の影響を示すため，データセットの時系列を考慮しない実験（交差検定による実験）とデータセットの時系列を考慮した実験を行い，結果を比較する.

機械学習アルゴリズムについては，既存手法の Support Vector Machine (SVM) に加え，Naive Bayes (NB)， k-Nearest Neighbor (kNN)，Decision Tree (DT)，Random Forest (RF) の計 5 種のアルゴリズムを使用した．また kNN の k の値については 1，3，5，7，9 の 5 パターンで実験を行った．交差検定の実験では，既存研究と同じく，5 分割交差検定とした.

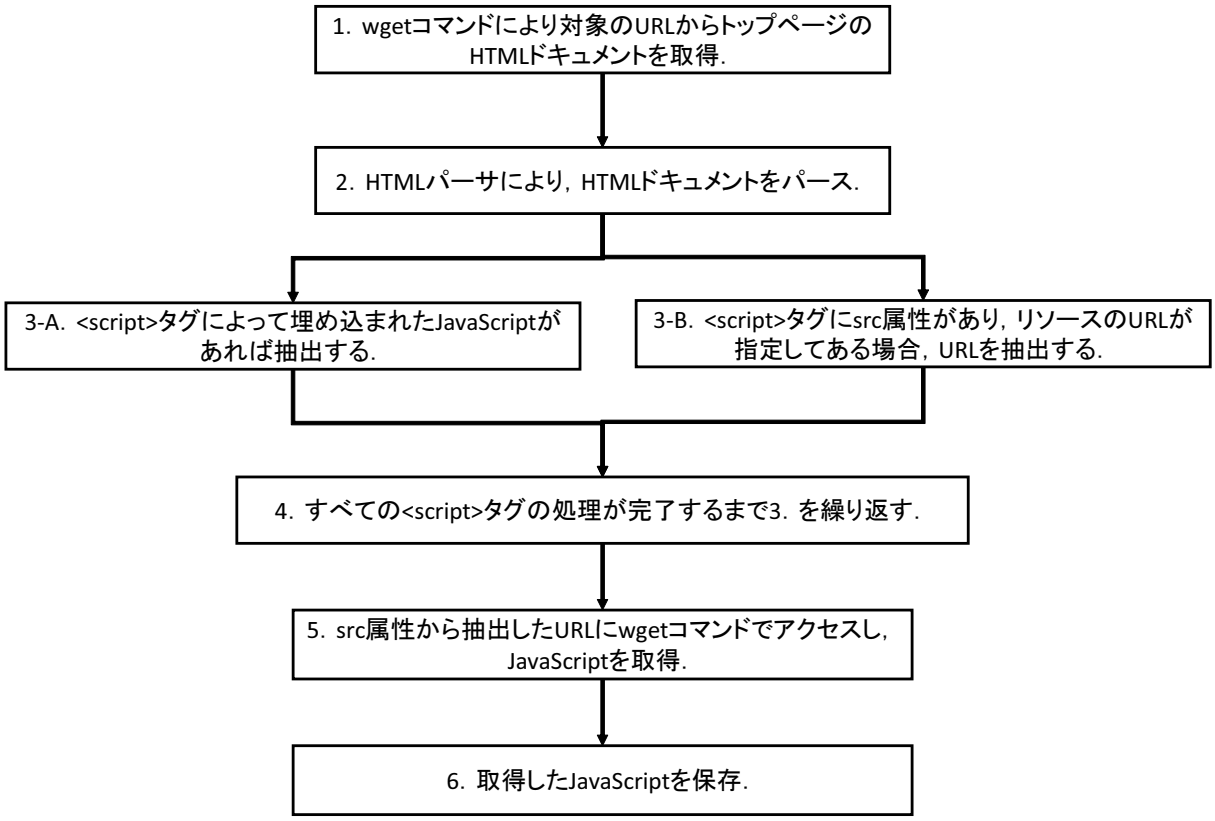

図 5.2: 良性 JavaScript データ収集の流れ

時系列を考慮した実験においては，既存研究 [9] の評価実験の問題点を解決するべく以下の 2 点について改善を行う．

5.3.1 学習データ・テストデータの分割について

データセットの時系列を考慮した実験においては，実世界の検知に準じて，古いデータを学習データ，新しいデータをテストデータとなるよう検証実験用 JavaScript データセットを分割する．学習用・テスト用データセットの内訳は，良性データの半分と 2011 年の悪性データが学習データ，良性データの残り半分と 2012 年から 2014 年の悪性データがテストデータである．

5.3.2 SVM のパラメータチューニングについて

SVM のカーネルは既存研究 [9] に則り RBF カーネルとし，C と γ の 2 つのパラメータはグリッドサーチによって最適と思われる値を求めた．但し，時系列を考慮した実験の場合は，実世界の評価になぞらえ，学習データのみを用いてグリッドサーチを行い，値を求

表 5.2: 時系列を考慮しない場合 (交差検定) の実験結果

		Accuracy	Precision	FNR	FPR
NB		0.9247	0.8369	0.1954	0.04276
SVM ($C = 329.9, \gamma = 61.7$)		0.9638	0.9226	0.09384	0.02064
DT		0.9548	0.8711	0.07394	0.03738
RF		0.9769	0.9822	0.09162	0.004485
kNN	k=1	0.9616	0.9159	0.09715	0.02243
	k=3	0.9546	0.9424	0.1612	0.01405
	k=5	0.9576	0.9602	0.1634	0.009569
	k=7	0.9572	0.9585	0.1645	0.009868
	k=9	0.9544	0.9578	0.1777	0.009868

表 5.3: 時系列を考慮した場合の実験結果

		Accuracy	Precision	FNR	FPR
NB		0.8960	0.8988	0.2618	0.03589
SVM ($C = 501.0, \gamma = 5.92$)		0.8480	0.9453	0.4737	0.01316
DT		0.7916	0.8608	0.6316	0.02572
RF		0.7414	0.9558	0.8504	0.002990
kNN	k=1	0.7385	0.8582	0.8407	0.01136
	k=3	0.7410	0.9180	0.8449	0.005981
	k=5	0.7544	0.9408	0.8019	0.005383
	k=7	0.7410	0.8984	0.8407	0.007775
	k=9	0.7410	0.8592	0.8310	0.01196

めた．また，グリッドサーチの際の評価指標は FNR（偽陰性率）を最重視し，パラメータの値を決定している．FNR が高い場合，多くの悪性データを良性と誤判断し，検知漏れが多いことを意味している．攻撃検知においては，良性を悪性と誤判断するより，悪性を良性と誤判断する方が危険であり，FNR の数値が重要であると考えられるため最重視した．

5.4 実験結果

時系列を考慮しない場合（交差検定）の実験結果を表 5.2 に，時系列を考慮した場合の実験結果を表 5.3 にそれぞれ示す．

5.5 考察

時系列を考慮しない実験（表 5.2）において，FNR は DT が最もよく，RF，SVM がそれに続く結果となった．Accuracy は RF が最もよく，次いで SVM が良い結果であった．この結果から，DT は，悪性検体の検知漏れ（偽陰性）は少ないが，偽陽性が SVM，RF よりも多く，RF および SVM は Accuracy が高く，全体的な性能は高いが DT に比べると悪性検体の検知漏れが多いことが分かる．これら 3 つのアルゴリズムは時系列を考慮しない実験（交差検定）においては，よい検知性能を示している．また，kNN については，k の値が増えるにつれ，FNR は悪化しており，悪性検体の検知漏れが多くなっている．NB に関しては，時系列を考慮しない実験において，FNR・Accuracy のどちらも他のアルゴリズムより悪い結果となった．

時系列を考慮した実験 (表 5.3) では，全体的にスコアは悪化している．特に FNR において RF と kNN は 0.8 (80%) を超えており，悪性検体の大部分を良性として誤判断している．交差検定で最も低い FNR を示した DT においても，FNR は 0.6316 であり過半数の悪性検体を誤判断している．また SVM においても，FNR は 0.4737 であり半数近くの悪性データを良性と誤判断し，正しく識別できていない．一方，Naive Bayes では，FNR は 0.2618 であり，約 74%の悪性データを悪性として正しく識別できている．また，Accuracy においても Naive Bayes は 0.8960 であり，他のアルゴリズムと比べ最も良い結果を示しており，全体的により良く識別できているといえる．Naive Bayes においては FNR が低い代わりに FPR が他のアルゴリズムよりも高くなっているが約 3.6%であり，先述のとおり良性を悪性と誤判断するより悪性を良性と誤判断する方が危険であるため，大きな問題ではないと考えられる．この結果から，実世界の検知においては，Naive Bayes が有効である可能性があるといえる．

また，既存手法 [9] の機械学習アルゴリズムである SVM における結果について，4.4 で述べたように，交差検定における SVM のパラメータのチューニングにはテストデータによるチューニングが内包されていたため交差検定による実験ではよい結果を示していた．しかし，時系列を考慮した実験の場合，実世界における検知になぞらえ，学習データのみから SVM のパラメータをチューニングしている．これにより，時系列を考慮した場合の SVM における評価スコアが下がったと考えられる．

第6章　bigramを用いた評価

6.1　bigramを用いた評価実験と実験結果

データセットの時系列を考慮した評価においては，手法の拡張として，bigramによる文字出現頻度を新たな特徴量としての実験も行った．実験に用いる学習データ・テストデータや，SVMのパラメータのチューニング方法は5.3の時系列を考慮した実験と同様である．

bigramにおいては，特徴ベクトルの次元が$94^2 = 8836$次元となり，膨大になるため次元の削減を行った．学習データの内，良性データと悪性データの間で出現頻度の差が大きい組の上位1767個を特徴として採用し，元の次元に比べ2割の次元（1767次元）に削減した場合と，上位4418個を特徴として採用し，元の次元に比べ5割の次元（4418次元）に削減した場合の2パターンの実験を行った．1767次元の特徴ベクトルを用いた実験結果を表6.1に，4418次元の特徴ベクトルを用いた実験結果を表6.2にそれぞれ示す．

6.2　考察

まず，1767次元の特徴ベクトルを用いた実験結果（表6.1）について考察する．unigram（表5.3）と比較し，NB，SVM，RFではFNRが悪化している．kNNでは，FNRは改善しているもののkの値によらず0.69 (69%) を超えており多くの検知漏れがあることが分かる．また，DTのFNRにおいても改善はしているが，1767次元の特徴ベクトルを用いた実験結果のうち最も低いFNRはNBの0.4571であり，表5.3におけるNBの結果である0.2618よりも悪い結果となった．

次に，4418次元の特徴ベクトルを用いた実験結果（表6.2）について考察する．unigram（表5.3）と比較し，NB，SVM，RFにおいてFNRが悪化している．kNNでは，FNRは改善しているもののkの値によらず0.72 (72%) を超えており，1767次元の特徴ベクトルを用いた実験と同様に多くの検知漏れがあることが分かる．また，DTのFNRにおいても改善はしているが，4418次元の特徴ベクトルを用いた実験結果のうち最も低いFNRはNBの0.4460であり，1767次元の場合と同様に，表5.3におけるNBの結果よりも悪い結果となった．

これらの結果から，bigramによる文字出現頻度は，良い特徴であるとはいえない．スコアが向上していない理由として，JavaScript中の文字間の関連性が低い可能性や，次元削減の方法が効果的ではない，などが考えられる．

表 6.1: bigram による特徴 (1767 次元) を用いた実験結果

		Accuracy	Precision	FNR	FPR
NB		0.8438	0.8991	0.4571	0.02632
SVM ($C = 60.0, \gamma = 5.0$)		0.8250	0.9749	0.5693	0.004785
DT		0.8396	0.9183	0.4861	0.01974
RF		0.7339	0.9670	0.8781	0.001794
kNN	k=1	0.7715	0.9187	0.7341	0.01017
	k=3	0.7853	0.9444	0.6939	0.007775
	k=5	0.7857	0.9563	0.6967	0.005981
	k=7	0.7853	0.9602	0.6994	0.005383
	k=9	0.7861	0.9605	0.6967	0.005383

表 6.2: bigram による特徴 (4418 次元) を用いた実験結果

		Accuracy	Precision	FNR	FPR
NB		0.8404	0.8696	0.4460	0.03589
SVM ($C = 216, \gamma = 2.3$)		0.7682	0.9563	0.7576	0.004785
DT		0.8371	0.9171	0.4945	0.01974
RF		0.7410	0.9636	0.8532	0.002392
kNN	k=1	0.7623	0.8964	0.7604	0.01196
	k=3	0.7740	0.9209	0.7258	0.01017
	k=5	0.7749	0.9463	0.7313	0.006579
	k=7	0.7753	0.9466	0.7299	0.006579
	k=9	0.7757	0.9426	0.7271	0.007177

また，1767 次元の特徴ベクトルを用いた実験結果と 4418 次元の特徴ベクトルを用いた実験結果を比較すると，次元を増やすことで NB と RF においては，FNR がわずかに改善されている．よって特徴ベクトルの次元を削減せず，$94^2 = 8836$ 次元の特徴ベクトルを用いれば，スコアが改善される可能性はあるが，計算コストが増えてしまい，コスト面でのメリットが薄くなってしまう．また，特徴ベクトルを算出する時間も多くかかることになり，実用性が下がると考えられる．

第7章　総括

DbD 攻撃における悪意のある難読化 JavaScript の検知手法として，文字出現頻度を特徴ベクトルとし，機械学習により検知を試みる既存手法 [9] が提案されている．この既存手法の評価においてはデータセットの時系列を考慮せずに交差検定が行われており，時系列的に道理に合わず，実世界に沿った評価ができていないと考えられる．そこで本稿では，この手法に対して，時系列を考慮した際の影響を示すことを目的とし，評価の厳密化を行った．時系列を考慮した際の影響を示すため，既存研究 [9] における評価方法である交差検定と，データセットの時系列に基づく実験を D3M dataset を用いて行い，実験結果の比較と考察を行った．

データセットの時系列に基づく評価の結果，交差検定の場合に比べ評価スコアが全体的に下がること，特に既存手法で用いられている SVM は Naive Bayes に比べスコアが低下することを示した．その結果，時系列に基づく評価の場合，つまり現実の検知においては Naive Bayes が有効である可能性があることが判明した．

また，手法の拡張として，bigram による文字出現頻度を新たな特徴量として実験を行った．しかし，評価スコアは向上しておらず，今回の実験では有効な特徴ではないことが判明した．

今回の時系列を考慮した実験において，Naive Bayes では，約 74%の悪性データを正しい識別できているが，さらに精度を上げるため，文字出現頻度以外の特徴量と組み合わせることを検討する必要がある．

第8章　対外発表

- 本田 仁, 面 和成, “Drive-by-Download 攻撃における難読化 JavaScript 検知手法についての考察”, The 33rd Symposium on Cryptography and Information Security (SCIS 2016), 2016.

謝辞

本研究を進めるにあたり，指導教員である面和成准教授からは，研究の方針についてのアドバイスや論文作成の際のご指摘など，多くのご指導を賜りました．心より御礼申し上げます．また，宮地充子教授には副指導教官としてご支援頂きました．ここに感謝の意を表します．さらに，ゼミなどにおいて様々な知識や提言を頂いた面研究室の皆様に感謝致します．

参考文献

[1] Takashi Adachi and Kazumasa Omote, “An Approach to Predict Drive-by-Download Attacks by Vulnerability Evaluation and Opcode”, The 10th Asia Joint Conference on Information Security (AsiaJCIS 2015) , 2015.

[2] “Alexa Top 500 Global Sites”, www.alexa.com/topsites

[3] Davide Canali, Marco Cova, Giovanni Vigna, Christopher Kruegel, “Prophiler: a fast filter for the large-scale detection of malicious web pages”, The 20th international conference on World wide web, 2011.

[4] HTMLParser, https://docs.python.org/2.7/library/htmlparser.html

[5] IBM, “2014 年下半期 Tokyo SOC 情報分析レポート”, www-935.ibm.com/services/jp/ja/it-services/soc-report/

[6] 情報処理推進機構, “情報セキュリティ 10 大脅威 2015”, www.ipa.go.jp/security/vuln/10threats2015.html

[7] G.K. Jayasinghe, J.S. Culpepper, P. Bertok, “Efficient and effective realtime prediction of drive-by download attacks”, Journal of Network and Computer Applications Volume 38, February 2014.

[8] P. Likarish, E. Jung, I. Jo “Obfuscated malicious javascript detection using classification techniques”, MALWARE ’09, 2009.

[9] 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”, 2014-CSEC-64, vol.21, pp. 1-7, 2014.

[10] Python, https://www.python.org/

[11] scikit-learn, http://scikit-learn.org/stable/

[12] W. Xu, F. Zhang, S. Zhu, “Jstill: Mostly static detection of obfuscated malicious javascript code”, CODASPY ’13, 2013.