情報処理学会論文誌 Vol.57 No.2 573–582 (Feb. 2016)

HTML5を用いた仮想Webブラウザの提案と評価

早川 智一[1,a)] 疋田 輝雄[1,b)]

受付日 2015年5月29日, 採録日 2015年9月2日

概要：本論文では，HTML5 と JavaScript 関連技術とを用いた仮想 Web ブラウザ（以下，仮想ブラウザ）の提案を行う．仮想ブラウザの目的は，悪意のあるコンテンツを含む Web ページ（以下，悪意のある Web ページ）から閲覧者を保護することにある．仮想ブラウザは，閲覧者が要求した Web ページを外見が等価な画像に透過的に変換することで，悪意のある Web ページの脅威を低減させる．仮想ブラウザのネットワーク転送量や応答時間の評価結果は，仮想ブラウザが閲覧者の利便性を大幅に低下させることなく悪意のある Web ページの脅威を低減できることを示した．我々は，この評価結果から，仮想ブラウザが悪意のある Web ページから閲覧者を保護する有用な手段たりうるという結論を得た．

キーワード：Web ブラウザ，HTML5，JavaScript，セキュリティ，プロキシ

Proposal and Evaluation of Virtual Web Browser by Using HTML5

Tomokazu Hayakawa[1,a)] Teruo Hikita[1,b)]

Received: May 29, 2015, Accepted: September 2, 2015

Abstract: This paper proposes a method of a virtual Web browser that uses HTML5 and JavaScript-related technologies only, which aims to provide users a way of safer Web-browsing environment that enables users to browse malicious Web pages without any harm. The virtual Web browser makes malicious Web pages harmless by transparently transforming Web pages requested by users into equivalent images. We evaluated bandwidths and response times of the virtual Web browser, and results show that the virtual Web browser realizes a safer Web-browing environment without sacrificing user experience. We conclude that the virtual Web browser is a great help to protect users from malicious Web pages.

Keywords: Web browser, HTML5, JavaScript, security, proxy

1. はじめに

1.1 背景

WWW（World Wide Web）の普及とともに，悪意のあるコンテンツ[*1]を含む Web ページ（以下，悪意のある Web ページ）が問題となっている．この顕著な例として，当該 Web ページを閲覧しただけで閲覧者にウイルスなどを感染させる「ドライブ・バイ・ダウンロード攻撃」[1] がある．また，最近では，ドライブ・バイ・ダウンロード攻撃を応用する「水飲み場型攻撃」[2] が，新しい脅威として認識されてきている．

一般に，悪意のある Web ページへの対策としては，

(1) 最新の Web ブラウザ（以下，ブラウザ）を使うこと，
(2) ブラウザのプラグイン（例：Java Applet・Flash・PDF など）を最新化すること，
(3) 最新のアンチウイルス製品を使うこと，
(4) 最新の OS（Operating System）を使うこと

などがあげられるが，これらの対策を講じるのが難しい場合や講じても不十分である場合が少なくない．たとえば，これらの対策は，バグの修正パッチやウイルスのパターンファイルが適用される前に行われるゼロ・デイ攻撃 [3], [4] には効果が薄い．また，企業などでは，前述の対策を次の

[1] 明治大学理工学部情報科学科
School of Science and Technology, Meiji University, Kawasaki, Kanagawa 214–8571, Japan
[a)] t_haya@cs.meiji.ac.jp
[b)] hikita@cs.meiji.ac.jp

[*1] 本論文では，「悪意のあるコンテンツ」とは，ウイルスやワームなどを広義に指すものとする．

理由からとらない（とれない）場合がある：
(1) 各種ソフトウェアを最新に維持し続けるためには様々な費用がかかる[*2]；
(2)（仮に費用を確保しても）社内システムの運用上の理由でソフトウェアを最新にできない[*3].

したがって，悪意のある Web ページへの対策が必要であるが，実運用環境に適用可能な実用性の高い方法は提案されていない（6 章）.

1.2 提案概要

我々は，悪意のある Web ページへの対策として，HTML5 [8] 準拠のブラウザ上で動作する「仮想 Web ブラウザ」（以下，仮想ブラウザ）[9], [10] を提案する（2 章）.

仮想ブラウザのアイデアは，
(1) 閲覧者のブラウザからの HTTP 要求を中継する際に，
(2) 要求された Web ページを外見が等価な画像に変換し，
(3) 変換した画像を HTTP 応答として閲覧者のブラウザに送り返す
ことで悪意のある Web ページの脅威を低減させ，閲覧者を保護する点にある（3 章）.

仮想ブラウザは，クライアント環境とサーバ環境とで構成される．仮想ブラウザのクライアント環境は HTML5 で記述され，HTML5 に準拠したブラウザ上で閲覧者から見て透過的に——あたかも仮想化されていないかのように——動作する．仮想ブラウザのサーバ環境は，認証・中継サーバと画像化サーバとで構成される.

我々は，仮想ブラウザを，HTML5 と JavaScript 関連技術のみを用いて実装した（4 章）．これは，実装に用いる技術を Web 開発の標準技術に限定することで，移植性を最大化しつつ開発・運用コストを低減させるためである.

我々は，仮想ブラウザのネットワーク転送量と応答時間とを評価し，仮想ブラウザが閲覧者の利便性を大幅に犠牲にすることなく悪意のある Web ページの脅威を低減できるという結論を得た（5 章）.

1.3 本論文の構成

2 章では，提案手法について説明する．3 章と 4 章では，提案手法の設計と実装について詳説する．5 章では，提案手法の評価結果について報告する．6 章では，本研究と関連する研究や技術について言及する．7 章では，まとめと今後の課題や展望について述べる.

[*2] ソフトウェアのライセンス料のほかに，ソフトウェアの更新を全社員に徹底させることで発生する人件費（それを回避するために専門の人員をかかえる場合にはその人件費）も考慮する必要がある.

[*3] たとえば，業務で使用しているアプリケーションの仕様や制約で，使用できる OS やブラウザが Windows XP や Internet Explorer 6 に制限される場合がある（各種の移行サービス [5], [6], [7] の存在がこのことを示唆している）.

2. 提案手法：仮想 Web ブラウザ

2.1 前提

我々は，仮想ブラウザの使用環境として，主に企業などの組織を前提とした（個人での使用も可能である）．これは，個人よりも組織の方が，悪意のある Web ページを閲覧した際の情報漏洩などのリスクが高いと考えるためである.

さらに，仮想ブラウザは，Web 閲覧時の安全性を高める既存の手法との併用を前提とした．これは次の理由による：
(1) 既存の手法と併用する方が，定性的に考えて安全性が向上する；
(2) 最近の攻撃は高度化・複雑化してきており，単一の手法で網羅的に防御することは困難である；
(3) 単一の手法で網羅的に防御しようとすると，その手法（およびその実装）に脆弱性があった場合に，閲覧者を守る術がなくなる.

既存の手法との併用の仕方にはいろいろな選択肢があるが，一例として以下のような運用を我々は想定している：
(1) 信用できる Web サイト（例：社内の業務アプリケーション）は，生のブラウザで直接アクセスする；
(2) 信用できない未知の Web サイトは，悪意のある Web ページと仮定して仮想ブラウザでアクセスする；
(3) 未知の Web サイトのうちシステム管理者などが信用できると判断したものは，次回からは信用できる Web サイトとして生のブラウザで直接アクセス可能にする.

この運用法により，信用できる Web サイトの閲覧時には利便性の低下はまったく起こらず，未知の Web サイトに対してのみ仮想ブラウザによる閲覧が行われるようになる.

2.2 目的

仮想ブラウザの目的は，実用性や利便性をなるべく低下させずに，悪意のある Web ページから閲覧者を保護することにある．なお，ここでの実用性とは，仮想ブラウザが
(1) 悪意のある Web ページの脅威を低減させる能力を提供し，
(2) 既存の環境に容易に導入でき，
(3) 低い初期・運用コストで使用できる
ことを意味するものとする．また，ここでの利便性とは，閲覧者が仮想ブラウザを使う際に
(1) 実行環境として，追加ソフトウェアをインストールすることなく日頃から使用しているブラウザを使え，
(2) 従来どおりブックマーク機能が使え，
(3) 従来どおりキーボードやマウスで操作でき，
(4) 従来どおりフォームの入力や送信ができ，
(5) Cookie や Local Storage などの永続化機能を使う Web アプリケーションも利用できる
ことを意味するものとする.

一方で，生のブラウザを完全に模倣・代替することは，仮想ブラウザの目的ではない．なぜならば，多くの機能を持っている生のブラウザを完全に仮想化することは，不可能ではないものの現実的ではないためである．

2.3 無害化手法

仮想ブラウザは，

(1) 閲覧者のブラウザからの HTTP 要求を中継し，
(2) その HTTP 要求に対する本来のHTTP 応答を，外見が等価な画像に隔離環境で透過的に変換し，
(3) 変換した画像を HTTP 応答として閲覧者のブラウザに返却して表示させる

ことで悪意のある Web ページの脅威を低減させる．

この手法により，利点と欠点との双方が生じる．利点の例としては，閲覧者は，仮想ブラウザを自身のブラウザ上で透過的に使えるようになる．欠点の例としては，閲覧者は，ファイルのアップロードやダウンロードができなくなる．この利点と欠点とは利便性とセキュリティとのトレードオフの関係にあり，運用方法を工夫（3.1 節）することで実用上の許容範囲に収まると我々は考える．

2.4 仮想 Web ブラウザにできること・できないこと

仮想ブラウザにできることは，Web の閲覧に関しては生のブラウザとおおむね同等である．これは，仮想ブラウザが

(1) 生のブラウザとほぼ同等の機能を持つブラウザ（Web エンジン）を内部で使っており，
(2) 生のブラウザが単一プロセス内で行っているイベントの発生と処理とをネットワーク越しに再現しているためである．同様の理由から，JavaScript の再現能力に関しても本質的には同等といえる．

一方で，画像化という手法ゆえの制限も存在する．具体的には，ブラウザの画面を頻繁に更新するような機能（例：`<canvas>` 要素や `<video>` 要素）を使った Web ページを表示しようとすると，短時間で大量の画面更新要求が発生し，ネットワーク転送量が大きくなってしまう．

別の制限として，仮想ブラウザが内部で画像化に使用するブラウザの種類によって Web ページの表示が崩れる可能性がある．これについては，生のブラウザを使っても発生する問題なので，画像化に使用するブラウザを柔軟に変更できるようにして対処する．

また，仮想ブラウザは，閲覧者の PC 上のハードウェアにアクセスすることはできない．これによって，音声の再生や Web カメラの利用はできなくなる．一方で，ハードウェアへアクセスできないことはセキュリティを向上させるため，利便性とセキュリティとのトレードオフである．

図 1 情報処理学会の Web ページを表示した仮想 Web ブラウザのスクリーンショット

Fig. 1 Screenshot of virtual web browser that shows IPSJ (Information Processing Society of Japan) web page.

2.5 仮想 Web ブラウザのスクリーンショット

図 1 は，情報処理学会の Web ページを表示した仮想ブラウザのスクリーンショットである（図中では Firefox[11] 上で実行）．注目すべき点として，図中のコンテキストメニューに「ページのソースを表示」が表示されていない点をあげる．これは，Web ページ全体が 1 枚の画像に変換されているためである．なお，仮想ブラウザは閲覧者の操作イベントを捕捉して処理するため，仮想ブラウザ上に表示されているリンクはクリック可能である．リンクがクリックされると，仮想ブラウザは新しい URL に遷移し，新しい URL に対応した画像を表示する．

3. 仮想 Web ブラウザの設計

3.1 前提

仮想ブラウザは，サーバとクライアントとから構成される．サーバはクライアントからの HTTP 要求に対する本来の HTTP 応答を画像化し，クライアントはサーバが画像化した HTTP 応答を表示する．サーバは，閲覧者の認証を行う「認証・中継サーバ」と Web ページの画像変換を行う「画像化サーバ」とで構成される．クライアントは HTML5 で実装された Web アプリケーションであり，HTML5 準拠のブラウザ上であれば OS を問わずに動作する．

図 2 に，我々が想定する仮想ブラウザのネットワーク構成の一例を示す．当該ネットワークは，

(1) 2 つのファイアウォールに挟まれた非武装地帯（DMZ：DeMilitarized Zone）を少なくとも 1 つ含み，
(2) 認証・中継サーバと画像化サーバとを DMZ に配置し，
(3) 閲覧者の PC を内部ネットワークに配置する

ことが望ましい．この構成により，仮想ブラウザを構成するサーバの一部または全部が悪意のある Web ページによって汚染された場合でも，閲覧者への被害を最小限に抑えることができる．これは，一般に，内部ネットワーク・DMZ

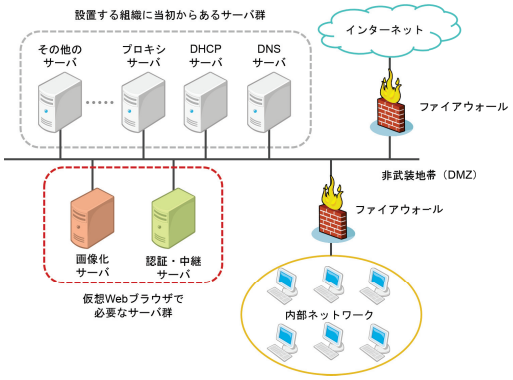

図 2 仮想 Web ブラウザのネットワーク構成
Fig. 2 Network structure of virtual web browser.

間のファイアウォールは，内部ネットワークから DMZ への通信は許可するが，その逆は拒否するためである（DMZ・インターネット間のファイアウォールもほぼ同様）．

我々は，仮想ブラウザを，既存のプロキシや他のセキュリティ製品と併用できるように設計した．具体的には，PAC（Proxy Auto Configuration）ファイル [12] を使用することを想定している．PAC ファイルを用いることで，閲覧者のブラウザに

(1) どの URL を直接的に閲覧させ（例：組織内部の業務アプリケーション），
(2) どの URL を既存のプロキシや他のセキュリティ製品を経由して閲覧させ（例：組織が認可した外部の Web サイト），
(3) どの URL を仮想ブラウザを経由して閲覧させるか（例：外部の未知の Web サイト）

をサーバ側で一括して指定することが可能になる*4．これにより，信用できない（疑わしい）Web ページのみ仮想ブラウザを経由して閲覧させることが可能になり，閲覧者の利便性の低下を最小限に抑えつつ安全性を向上させることができる．

3.2 仮想 Web ブラウザの動作

図 3 に仮想ブラウザの動作の概要を，**図 4**，**図 5** に仮想ブラウザの動作の詳細をそれぞれ示す．これらの図が示す仮想ブラウザの動作の流れは次のとおりである：

(1) 閲覧者のブラウザが HTTP 要求を発行する（例：URL を入力する，ブックマークを選択する，リンクをクリックするなど）；
(2) 最初の HTTP 要求に対して，認証・中継サーバがダイジェスト認証 [13] を要求して，ユーザ名とパスワードとを問い合わせる（この認証情報は，画像化サーバ内での画像化プロセスの識別や隔離に使用される）；

*4 組織内の全ブラウザのプロキシ設定を手動で行うことは現実的ではないため，多くの組織がこの方法を——あるいは PAC ファイルの指定すらも省略するためにブラウザの「プロキシの自動検出」機能を使う方法を——採用しているはずである．

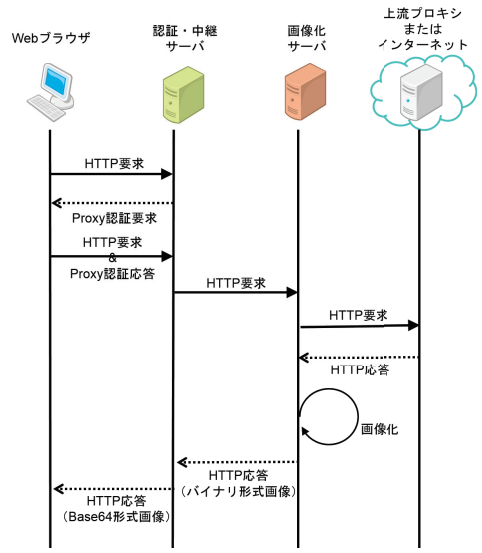

図 3 仮想 Web ブラウザの動作概要
Fig. 3 Behavioral overview of virtual web browser.

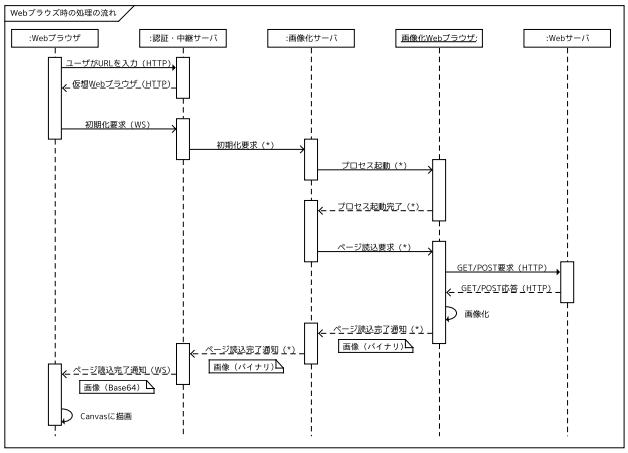

図 4 仮想 Web ブラウザのシーケンス図（Web ブラウジング時）
Fig. 4 Sequence diagram of virtual web browser (when browsing web pages).

(3) ブラウザは閲覧者に認証情報を入力させ，認証情報を含む HTTP 要求を再発行する；
(4) 画像化サーバは，閲覧者ごとに専用の画像化プロセスを認証情報に基づいて隔離した環境で起動する；
(5) 認証・中継サーバは，HTML5 で実装された仮想ブラウザを閲覧者のブラウザに返す；
(6) 仮想ブラウザは，WebSocket を使って HTTP 要求を再発行する；
(7) 認証・中継サーバは，画像化サーバに HTTP 要求を転送する；
(8) 画像化サーバは，HTTP 要求を本来の宛先サーバまたは上流プロキシに転送する；
(9) 画像化サーバは，HTTP 要求を転送したサーバから HTTP 応答を受け取る；
(10) 画像化サーバは，受け取った HTTP 応答を隔離した環境で外見が等価な画像に変換する；

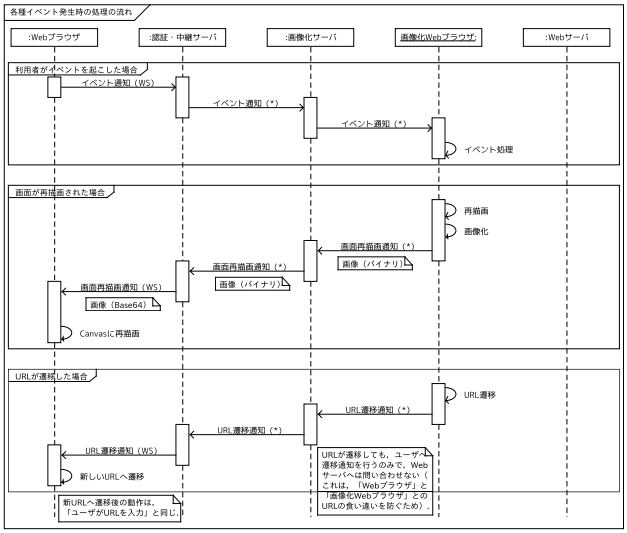

図 5 仮想 Web ブラウザのシーケンス図（イベント発生時）
Fig. 5 Sequence diagram of virtual web browser (when events fired).

(11)画像化サーバは，変換した画像を認証・中継サーバにバイナリ形式で返す；
(12)認証・中継サーバは，受け取ったバイナリ形式の画像を Base64 [14] 形式に変換して仮想ブラウザに返す；
(13)仮想ブラウザは，受け取った Base64 形式の画像を `<canvas>` 要素上に表示する．

ここで重要なことは，HTTP の代わりに WebSocket を用いて認証・中継サーバと仮想ブラウザとが通信する点である．これは，WebSocket が HTTP とは異なり双方向・非同期に通信する機能を備えるためであり，画像化サーバ内の画像化プロセスと仮想ブラウザとの同期を可能にする．たとえば，画像化プロセス内で読み込んだ Web ページが新しい URL に遷移した場合，当該プロセスは仮想ブラウザに URL の遷移通知を行い，これによって仮想ブラウザも新しい URL へと遷移する．仮に，WebSocket の代わりに HTTP を使うと，このような双方向・非同期の通信が困難になる[*5]．別の例として，仮想ブラウザがサポートする JavaScript イベント（`onresize`，`onscroll`，`onmouseup`，`onmousedown`，`onmousemove`，`onclick`，`ondblclick`，`onkeyup`，`onkeydown`，`onkeypress` など）が発生した場合には，仮想ブラウザは WebSocket を通じて画像化サーバ内の画像化プロセスにイベント情報を通知する．これにより，仮想ブラウザが，フォーム入力・リンクのクリック・画面のスクロール・画面のサイズ変更などのユーザイベントを捕捉して処理することが可能になる．

3.3 画像化プロセスの隔離

画像化プロセスを隔離するためには仮想化技術（6.1 節）を用いるのが一般的であるが，我々は，安価で軽量な変換を実現するために，Linux の API（Application Programming Interface）を用いる方法を選択した．具体的には，画像化プロセスの特権を放棄するために `setuid(2)` と `setgid(2)` とを使用し，画像化プロセスが汚染された場合の影響範囲を特定のディレクトリ下に閉じ込めるために `chroot(2)` を用いた．これらの（もしくは類似の）API は多くの Unix 系 OS で動作するため，仮想ブラウザのサーバ環境が多くの Unix 系 OS 上で稼働することが期待できる．

また，画像化プロセスの隔離によってスケーラビリティの向上も容易になる．なぜならば，1 人の閲覧者ごとに 1 つの画像化プロセスが隔離環境で起動し画像化プロセス間にはいっさいの依存関係が存在しないため，画像化サーバの処理能力が不足した場合に複数の画像化サーバに画像化プロセスを分散させることが容易にできるためである．

3.4 設計上の制限

仮想ブラウザには設計上の制限が存在する．以下に，それらの制限について述べる．一方で，これらの制限については，前述の PAC ファイルを用いた運用（3.1 節）をすることで緩和が可能であるため，実用上の差し支えは少ないと我々は考える（仮想ブラウザが使用されるのは，あくまでも疑わしい Web ページに対してのみであるため）．

3.4.1 HTTPS

仮想ブラウザは，HTTPS（HTTP over SSL/TLS）には対応していない．これは，仮想ブラウザの挙動がブラウザからは中間者（MITM：Man-In-The-Middle）攻撃と見なされ阻止されるためである[*6]．

3.4.2 ウィンドウのポップアップ

仮想ブラウザは，JavaScript を用いたウィンドウのポップアップには対応していない（ウィンドウが重なった状態で表示することは可能だが，前面のウィンドウに隠された背面のウィンドウの内容が見えなくなってしまう）．

4. 仮想 Web ブラウザの実装

4.1 ソフトウェア

我々は，仮想ブラウザを**表 1** のソフトウェアを用いて実装した．特徴として，プログラミング言語に JavaScript を一貫して採用した点をあげる．これは，JavaScript が Web 開発の事実上の標準言語であるため，クライアントとサーバとで同じ言語を使用することで，開発・保守のコストを低減できるためである．jQuery [15] は JavaScript の汎用ライブラリであり，仮想ブラウザにブラウザ間の互換性を持たせるために使用した．PhantomJS [16] は JavaScript で命令を記述できる WebKit [17] ベースの Headless[*7]なブ

[*5] 厳密には，Ajax や COMET などを使えば近い機能を実現することはできるが，WebSocket と比較すると非効率的である．
[*6] 厳密には，独自証明書を発行したり警告を無視したりすれば仮想ブラウザは動作するが，接続先サーバの真正性が確認できなくなるデメリットの方が大きいと我々は判断した．
[*7] GUI を必要とせずに動作するソフトウェア．GUI を必要とするソフトウェアよりも必要なリソースが少ないなどの利点がある．

表 1 仮想 Web ブラウザの実装に使用したソフトウェア
Table 1 Software for virtual web browser implementation.

ソフトウェア	バージョン	備考
jQuery	2.1.4	クライアント部分の実装
PhantomJS	2.0	画像化サーバの実装
Node.js	0.12.4	認証・中継サーバの実装
Apache HTTP Server	2.4.12	各種プロキシの実装
CentOS	7.1	OS

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>仮想Webブラウザ</title>
    <style type="text/css">
      * {
        margin: 0px;
        padding: 0px;
      } /* 画面上の余白を除去する */
    </style>
    <script type="text/javascript">
    /*
     * (1)jQueryを読み込む.
     * (2)各種イベントハンドラを登録する.
     * (3)認証・中継サーバと通信するための
     *    WebSocketを開く.
     * (4)WebSocketを通じてイベント情報を
     *    送受信する.
     */
    </script>
  </head>
  <body>
    <!-- Web画面表示用の唯一の要素 -->
    <canvas id="ui">
  </body>
</html>
```

図 6 仮想 Web ブラウザのスケルトン（HTML5）
Fig. 6 Skeleton of virtual web browser (HTML5).

ラウザであり，画像化サーバを実装するために使用した．Node.js [18] は JavaScript で命令を記述できるサーバサイドの実行環境であり，画像化サーバとやりとりする認証・中継サーバを実装するために使用した．Apache HTTP Server は，ダイジェスト認証が可能な HTTP/WebSocket プロキシとして使用した．

4.2 仮想 Web ブラウザのスケルトン

図 6 に，仮想ブラウザのスケルトンを示す．図より，仮想ブラウザは HTML の要素に `<canvas>` しか持たないことが分かる．この要素上に，画像化された Web ページが表示される．閲覧者の操作によって発生するイベントは，`<script>` 要素にあらかじめ組み込まれたイベントハンドラによって捕捉され，WebSocket を通じて画像化サーバ上の画像化プロセスに通知される．同様に，画像化プロセス上で発生したイベントは，WebSocket を通じて仮想ブラウザに通知され，あらかじめ定められた手続きに従って処理される．これらの動作により，仮想ブラウザはあたかも仮想化されていない生のブラウザであるかのように振る舞う．

表 2 Web ブラウザがサポートする HTTP 1.1 の圧縮形式
Table 2 HTTP 1.1 compression formats supported by modern web browsers.

Web ブラウザ[1]	バージョン	gzip	compress	deflate
Internet Explorer	11.0	○	×	○
Firefox	38.0	○	×	○
Chrome	43.0	○	×	○
Opera	29.0	○	×	○

[1] Safari は，JVN（Japan Vulnerability Notes）が Windows 版の使用中止勧告 [21] をしているため評価の対象外とした．

4.3 ネットワーク転送量の削減

我々は，ネットワーク転送量を削減するために，仮想ブラウザと認証・中継サーバとの間の通信を圧縮するように実装した．**表 2** に，HTTP 1.1 [19] で定められた圧縮形式とブラウザの対応状況とを示す．表より deflate か gzip かが選択候補となるが，我々は deflate を選択した．これは，deflate が，HTTP 1.1 だけでなく WebSocket の圧縮形式としてもサポートされているためである [20]．

4.4 実装上の制限

仮想ブラウザには実装上の制限が存在する．これらの制限は，我々が仮想ブラウザの実装に使用している PhantomJS（または PhantomJS が内部で使用しているフレームワークの Qt [22] や，Qt が内部で使用している WebKit）による部分が大きい．したがって，これらを独自の実装で置き換えれば，制限を回避することが可能である．

4.4.1 画像形式

仮想ブラウザが使用できる画像形式は，PNG と JPG のみである．仮想ブラウザは標準で PNG を使用するが，必要に応じて JPG に変更することも可能である．一般に，ファイルサイズは PNG>JPG となるが，JPG は非可逆圧縮で画像の劣化がある点に留意する必要がある．

4.4.2 プラグイン

仮想ブラウザは，閲覧に専用プラグイン（例：Java Applet・PDF・Flash など）を必要とする Web ページを画像化することはできない．一方で，専用プラグインを動作させることは画像化プロセスの汚染リスクを高めることになるので，利便性とセキュリティとのトレードオフである．

5. 評価

我々は，仮想ブラウザの有用性を評価するために 2 つの試験を行った．評価においては，Web 上のトラフィック量 [23] の上位 10 サイト（**表 3**）を使用した．

5.1 ネットワーク転送量による評価

第 1 に，仮想ブラウザと認証・中継サーバとの間のネットワーク転送量の測定を行った．評価対象には表 3 のサイトを使用し，オリジナルの Web ページ（当該 Web ページ

に含まれる CSS や JavaScript や画像などの外部コンテンツも含む）のサイズに対し，当該 Web ページを画像に変換した場合の画像形式および圧縮形式ごとのファイルサイズ（**表 4**）を評価値とした（評価値は 10 回の試行の平均値）．

評価の結果，仮想ブラウザは表 3 のすべての Web サイトを画像化することができた．また，表 4 から，deflate で圧縮することで，Base64 形式への変換で増加する画像サイズを変換前のサイズとほぼ同じにできることが分かる．

我々は，この評価の結果から，CPU 資源さえ十分ならば，仮想ブラウザがネットワーク転送量を増加させることなく Web ページを外見が等価な画像に変換できるという結論を得た．

表 3 Web 上のトラフィック量の上位 10 サイト
Table 3 Top 10 web sites ordered by traffic volume.

順位	Web サイト名	URL
1	Google.com	http://google.com/
2	Facebook.com	http://facebook.com/
3	Youtube.com	http://youtube.com/
4	Yahoo.com	http://yahoo.com/
5	Baidu.com	http://baidu.com/
6	Amazon.com	http://amazon.com/
7	Wikipedia.org	http://wikipedia.org/
8	Taobao.com	http://taobao.com/
9	Qq.com	http://qq.com/
10	Twitter.com	http://twitter.com

5.2 応答時間による評価

第 2 に，仮想ブラウザの応答時間を測定した．評価対象には表 3 のサイトを使用し，仮想ブラウザを使用した際に発生する遅延時間と画像変換にかかる時間とを評価値とした（評価値は 10 回の試行の平均値）．

評価の結果を**表 5** に示す．この中で「読込時間」は特に重要である．なぜならば，この値は，生のブラウザを使った場合と仮想ブラウザを使った場合との体感時間の差を表しているからである．ここで，体感時間の差が生じる理由は，生のブラウザはトップページを受け取ると画面の描画

表 4 表 3 のトップページの画像形式ごとのファイルサイズ（バイト）
Table 4 Image sizes (bytes) of top pages shown in Table 3.

Web サイト	生サイズ*1	JPEG	JPEG Base64	JPEG Base64 Deflate	PNG	PNG Base64	PNG Base64 Deflate
Google.com	282,949	22,910	30,548	17,986	36,669	48,892	35,156
Facebook.com	1,841,983	86,045	114,728	68,634	197,519	263,360	194,507
Youtube.com	2,574,599	352,287	469,717	305,638	1,384,083	1,845,446	1,372,784
Yahoo.com	1,571,936	455,125	610,501	403,917	1,132,177	1,504,048	1,113,362
Baidu.com	422,609	23,152	30,872	14,281	27,435	36,580	23,946
Amazon.com	5,582,703	428,599	571,466	391,707	2,265,560	3,020,748	2,253,555
Wikipedia.org	122,410	137,928	183,904	131,014	281,168	374,892	276,862
Taobao.com	2,427,956	172,384	229,848	141,732	915,262	1,220,351	910,249
Qq.com	3,620,629	913,987	1,218,650	856,511	3,235,599	4,314,134	3,215,905
Twitter.com	1,448,727	105,838	141,119	86,574	447,561	596,750	443,572

1 当該 Web サイトのトップページに含まれるすべてのコンテンツの合計サイズ．

表 5 表 3 のトップページに対する各種応答時間（ミリ秒）
Table 5 Response times (msec) for browsing top pages shown in Table 3.

Web サイト	外部リソース数*1	読込開始時間*2（a）	読込完了時間*3（b）	読込時間*4（$b-a$）	JPEG 変換時間	PNG 変換時間
Google.com	10	20	164	144	19	26
Facebook.com	34	190	1,376	1,186	54	78
Youtube.com	86	99	1,257	1,157	340	561
Yahoo.com	39	365	1,819	1,454	107	291
Baidu.com	12	248	367	119	28	33
Amazon.com	200	205	3,087	2,881	162	564
Wikipedia.org	20	129	662	532	42	70
Taobao.com	193	242	2,546	2,303	115	286
Qq.com	221	33	3,222	3,188	192	597
Twitter.com	16	136	920	783	72	171

1 当該 Web サイトのトップページから読み込まれている外部リソース（例：画像ファイル・CSS ファイル・JavaScript ファイル）の数．
2 当該 Web サイトのトップページがサーバから読み込まれるまでの時間（小数点以下切捨）．
3 当該 Web サイトのトップページに含まれるすべての外部リソースが読み込まれるまでの時間（小数点以下切捨）．
4 当該 Web サイトのトップページが読み込まれ始めてから完全に表示されるまでの時間（小数点以下切捨）．

を行いながら外部リソースを要求するのに対して，仮想ブラウザは画像化を行うという特性上すべての外部リソースの取得が完了するまでは画面の描画を開始できないためである．

表 5 より以下のことが分かる．多くの場合，
(1) 画像の変換時間は，最大でも 600 ミリ秒弱である；
(2) ページの読込完了時間は，最大でも 3 秒強である；
(3) 体感的な待ち時間の増加は，最大でも 3 秒弱である．

我々は，この評価の結果から，多くの場合において，仮想ブラウザが応答時間を大幅に増加させることなく Web ページを外見が等価な画像に変換できるという結論を得た．

5.3 セキュリティに関する考察

仮想ブラウザは，悪意のある Web ページを画像化することでその脅威を低減させる．ここで，画像化が脅威の低減にどの程度有効かを定量的に評価することは難しい．しかし，画像化による無害化は商用製品で採用されている例もあり（6.2 節），定性的には有効な方法であると我々は考える．

次に，仮想ブラウザが脅威を低減させきれない場合を考える．たとえば，閲覧者の PC が文献 [24] のような状況下において攻撃者が仮想ブラウザのサーバを乗っ取ることができれば，悪意のあるコンテンツを画像に埋め込んで閲覧者に実行させることは不可能ではない．このような攻撃に対しては，(1) 画像化サーバ上の重要なファイルのチェックサムを定期的に検査して改ざんを迅速に検出する，(2) Docker [25] などの既存の仮想化技術を用いて，汚染された環境の迅速な破棄・正常化を可能にする——などの手法と併用することでリスクの低減が可能である．

6. 関連研究・関連技術

6.1 仮想化技術

安全な Web ブラウジングを実現する技術の 1 つとして仮想化技術がある．仮想化技術には，大別してアプリケーション仮想化・デスクトップ仮想化・OS 仮想化がある．いずれの仮想化技術に対しても，ほとんどの仮想化技術が専用のソフトウェアや OS を必要とするのに対し，仮想ブラウザは HTML5 準拠のブラウザのみを必要とする点で異なる．これにより，追加のコストやソフトウェアを必要とせずに，低コストで安全な Web ブラウジングが可能になる．

6.2 その他

ネットエージェント株式会社の製品「防人」[26] は，標的型メール攻撃を防御するための製品である．このソフトはメール中継サーバとして動作し，メールの添付ファイルを画像化することで，閲覧者の PC が悪意のある添付ファイルで汚染されることを防ぐ．防人と仮想ブラウザとは，悪意のあるコンテンツを画像化して無害化する点に共通点があるが，その対象が異なる．防人はメールの添付ファイルを対象とするが，仮想ブラウザは Web ページを対象とする[*8]．

6.3 先行研究・類似研究

Palanques ら [27] は，“Secure Cloud Browser” という手法を提案している．彼らの研究と仮想ブラウザとは，要求された Web ページを隔離した環境で画像化し閲覧者を悪意のあるコンテンツから保護する点に類似性がある．一方で，彼らの手法は，攻撃者が閲覧者の PC 上で管理者権限を持っていることを前提としており，実行環境として Web ブラウザの他に JRE（Java Runtime Environment）を必要とする点で異なる．仮想ブラウザは，閲覧者を悪意のあるコンテンツから保護することを目的としており，HTML5 準拠のブラウザのみで動作する．

Grier ら [28] は，“OP web browser” という手法を提案している．彼らの研究と仮想ブラウザとは，要求された Web ページを隔離したプロセス内で画像化して閲覧者に返却する点に類似性があるが，彼らの研究は彼らが実装した専用のブラウザと JRE とを必要とする一方で，仮想ブラウザは HTML5 準拠のブラウザのみで動作する点で異なる．

Wang ら [29] は，“SafeFox” という手法を提案している．彼らの研究と仮想ブラウザとは，仮想環境でブラウザを動作させることで閲覧者を保護する点に類似性がある．一方で，彼らの手法はプロセスの保護に専用の仮想環境を必要とするが，仮想ブラウザは専用の仮想環境を必要とせずにプロセスの保護を実現している点で異なる．

7. おわりに

本論文では，悪意のある Web ページの脅威を低減させる手法として仮想ブラウザを提案した．評価の結果は，提案手法が閲覧者の利便性を大幅に犠牲にすることなく悪意のある Web ページの脅威を低減できることを示した．この結果から，提案手法が閲覧者の PC のセキュリティの強化に有用であり，悪意のある Web ページの脅威を低減させるための 1 つの解決策たりうるという結論を我々は得た．

今後の予定として，我々は，仮想ブラウザの詳細な評価と利便性・機能性の向上とを計画している．評価の具体例としては，仮想ブラウザが閲覧者の利便性を大幅に犠牲にしていないことを確認するために，複数の被験者を対象としたユーザビリティテストを行う予定である．利便性・機能性の向上の具体例としては，(1) 複数の領域に分割して画像化した Web ページを `<canvas>` 要素上でつなぎ合わせることによる高速化，(2) Web Workers [30] を用いた処

[*8] 一般に，メールの添付ファイルは他のコンテンツへのリンクを含まない静的コンテンツであるが，Web ページは他のコンテンツへのリンクを含む動的コンテンツである．よって，両者は本質的に異なるものであり，画像化にも異なる工夫や配慮が必要になる．

理の並列化，(3) レスポンシブ Web デザインへの対応強化——などを検討している．

謝辞 本研究は JSPS 科研費 26730041 の助成を受けたものです．

参考文献

[1] 独立行政法人情報処理推進機構：コンピュータウイルス・不正アクセスの届出状況 [2010 年 11 月分] について，独立行政法人情報処理推進機構（オンライン），入手先 ⟨http://www.ipa.go.jp/security/txt/2010/12outline.html⟩（参照 2015-05-05）．

[2] 株式会社ラック：日本における水飲み場型攻撃に関する注意喚起，株式会社ラック（オンライン），入手先 ⟨http://www.lac.co.jp/security/alert/2013/10/09_alert_01.html⟩（参照 2013-10-20）．

[3] 独立行政法人情報処理推進機構：コンピュータウイルス・不正アクセスの届出状況 [2009 年 7 月分] について，独立行政法人情報処理推進機構（オンライン），入手先 ⟨http://www.ipa.go.jp/security/txt/2009/08outline.html⟩（参照 2015-05-05）．

[4] 独立行政法人情報処理推進機構：修正プログラム提供前の脆弱性を悪用したゼロデイ攻撃について，独立行政法人情報処理推進機構（オンライン），入手先 ⟨http://www.ipa.go.jp/security/virus/zda.html⟩（参照 2015-05-05）．

[5] Symantec Corp.: Windows7 用の仮想化 Internet Explorer6，Symantec Corp.（オンライン），入手先 ⟨http://www.symantec.com/ja/jp/articles/article.jsp?aid=20100802_virtualized_internet_explorer6_for_windows7⟩（参照 2015-05-05）．

[6] 東芝情報機器株式会社：Windows XP 移行サービスメニュー，東芝情報機器株式会社（オンライン），入手先 ⟨http://www.toshiba-tie.co.jp/solution/data_migration/⟩（参照 2015-05-05）．

[7] 双日システムズ：アプリケーション仮想化の専門組織を立ち上げ，Windows XP からの移行支援サービスに本格参入，双日システムズ（オンライン），入手先 ⟨https://security.sojitz-sys.com/virtualization/news/pdf/20130510.pdf⟩（参照 2013-07-24）．

[8] W3C: HTML5, W3C (online), available from ⟨http://www.w3.org/TR/html5/⟩ (accessed 2015-05-05).

[9] Hayakawa, T. and Hikita, T.: Proposal for Virtual Web Browser by Using HTML5, *Proc. 5th FTRA International Conference on Computer Science and its Applications* (*CSA-13*), Danang, pp.225–232 (2013).

[10] 早川智一，疋田輝雄：HTML5 を用いた仮想 Web ブラウザの設計と実装，マルチメディア通信と分散処理ワークショップ論文集，pp.284–291 (2013).

[11] Mozilla Foundation: Mozilla Firefox, Mozilla Foundation (online), available from ⟨https://www.firefox.com/⟩ (accessed 2015-09-10).

[12] Microsoft: Chapter 26 – Using Automatic Configuration, Automatic Proxy, and Automatic Detection, Microsoft (online), available from ⟨http://technet.microsoft.com/library/Dd361918⟩ (accessed 2015-05-05).

[13] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A. and Stewart, L.: HTTP Authentication: Basic and Digest Access Authentication, The Internet Society (online), available from ⟨http://tools.ietf.org/html/rfc2617⟩ (accessed 2015-05-05).

[14] Josefsson, S.: The Base16, Base32, and Base64 Data Encodings, The Internet Society (online), available from ⟨http://tools.ietf.org/html/rfc4648⟩ (accessed 2015-05-05).

[15] jQuery Foundation: jQuery: The Write Less, Do More, JavaScript Library, jQuery Foundation (online), available from ⟨http://jquery.com/⟩ (accessed 2015-05-05).

[16] Hidayat, A.: PhantomJS: Headless WebKit with JavaScript API, PhantomJS.org (online), available from ⟨http://phantomjs.org/⟩ (accessed 2015-05-05).

[17] WebKit Team: The WebKit Open Source Project, WebKit Team (online), available from ⟨http://www.webkit.org/⟩ (accessed 2015-05-05).

[18] Joyent, Inc.: node.js, Joyent, Inc. (online), available from ⟨http://nodejs.org/⟩ (accessed 2015-05-05).

[19] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and Berners-Lee, T.: Hypertext Transfer Protocol – HTTP/1.1, The Internet Engineering Task Force (IETF) (online), available from ⟨http://www.ietf.org/rfc/rfc2616.txt⟩ (accessed 2015-05-05).

[20] Yoshino, T. and Google, Inc.: Compression Extensions for WebSocket draft-ietf-hybi-permessage-compression-21, The Internet Engineering Task Force (IETF) (online), available from ⟨https://tools.ietf.org/html/draft-ietf-hybi-permessage-compression-21⟩ (accessed 2015-05-05).

[21] JPCERT/CC and IPA：JVN#42676559：Safari においてリモートからローカルファイルを読み取り可能な脆弱性，JPCERT/CC and IPA（オンライン），入手先 ⟨http://jvn.jp/jp/JVN42676559/index.html⟩（参照 2015-05-05）．

[22] Digia Plc: Qt Project, Digia Plc (online), available from ⟨http://www.qt.io/developers/⟩ (accessed 2015-05-05).

[23] Alexa, Inc.: Alexa Top 500 Global Sites, Alexa, Inc. (online), available from ⟨http://www.alexa.com/topsites⟩ (accessed 2015-05-05).

[24] Microsoft：Microsoft Security Bulletin MS04-028：JPEG 処理 (GDI+) のバッファオーバーランにより、コードが実行される (833987)，Microsoft（オンライン），入手先 ⟨http://technet.microsoft.com/ja-jp/security/bulletin/ms04-028⟩（参照 2015-05-05）．

[25] Docker, Inc.: Docker – Build, Ship, and Run Any App, Anywhere, Docker, Inc. (online), available from ⟨https://www.docker.com/⟩ (accessed 2015-05-05).

[26] ネットエージェント株式会社：標的型メール攻撃の無害化対策「防人」，ネットエージェント株式会社（オンライン），入手先 ⟨http://www.netagent.co.jp/product/sakimori/⟩（参照 2015-05-05）．

[27] Palanques, M., Dipietro, R., del Ojo, C., Malet, M., Marino, M. and Felguera, T.: Secure Cloud Browser: Model and Architecture to Support Secure WEB Navigation, *Proc. 31st IEEE Symposium on Reliable Distributed Systems* (*SRDS*), Irvine, pp.402–403 (2012).

[28] Grier, C., Tang, S. and King, S.: Secure Web Browsing with the OP Web Browser, *Proc. IEEE Symposium on Security and Privacy*, Oakland, pp.402–416 (2008).

[29] Wang, J., Huang, Y. and Ghosh, A.: SafeFox: A Safe Lightweight Virtual Browsing Environment, *Proc. 43rd Hawaii International Conference on System Sciences* (*HICSS*), Honolulu, pp.1–10 (2010).

[30] W3C: Web Workers, W3C (online), available from ⟨http://www.w3.org/TR/workers/⟩ (accessed 2015-09-10).

早川 智一 （正会員）

1982 年生．2004 年明治大学理工学部情報科学科卒業．2007 年同大学大学院修士課程修了．2007 年（株）ティージー情報ネットワーク入社．2010 年明治大学理工学部助手．2013 年同大大学院博士課程修了．博士（工学）．2013 年同大学博士研究員．2014 年より同大学助教．ソフトウェアやウェブ技術等に関する研究に従事．2012 年度情報処理学会山下記念研究賞受賞．

疋田 輝雄 （正会員）

1947 年生．1970 年東京大学卒業, 1978 年理学博士．1989 年から明治大学情報科学科教授．計算理論，ウェブ技術等に興味を持っている．著書は『コンパイラの理論と実現』（共立出版）ほか,『情報科学こんせぷつ』（朝倉書店）編集．