HTML5を用いた仮想Webブラウザの設計と実装

早川 智一[1,a)]　疋田 輝雄[1,b)]

概要：本論文では，HTML5 と JavaScript 関連技術とを用いた仮想 Web ブラウザの提案を行う．仮想 Web ブラウザの目的は，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 Web ブラウザは，閲覧者の Web ブラウザがリクエストした 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，悪意のあるコンテンツを無害化する．我々は，仮想 Web ブラウザの通信量を評価し，仮想 Web ブラウザが画像化によって通信量を増加させないことを確認した．我々は，評価の結果から，仮想 Web 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え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

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」[20] がある [*2]．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」[18] 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) ブラウザの最新のプラグイン（例：Java Applet・Flash・PDF など）を使うこと，
(3) 最新のアンチウイルスを使うこと，
(4) 最新の OS（Operating System）を使うこと

などが挙げられるが，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 [19,22] には効果が薄い．また，企業などでは，前述の対策を次の理由から取らない（取れない）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 [*3]；
(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 [*4]．

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，実運用環境に適用が可能な実用性の高い方法は，あまり提案されていないようである（6 節）．

1.2 提案概要

我々は，悪意のある Web ページ（特にドライブ・バイ・ダウンロード攻撃）への対策として，HTML5 [15] 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）を提案する（2 節）．

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) 変換済みの画像を 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 ページを無害化し，閲覧者がウイルスなどに感染することを防ぐ点にある（3 節）．

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部分は 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

[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．
[*2] 近年では，Gumblar [7,21] が猛威を振るったことが記憶に新しい．
[*3] ソフトウェアのライセンス料の他に，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抱える場合にはその人件費）をも考慮する必要がある．
[*4] 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 [14,23,25] の存在が，このことを示唆している）．

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部分は，HTTP/WebSocket プロキシ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 関連技術とを用いて実装した（4 節）．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・運用コストを低減するためである．

我々は，仮想ブラウザの通信量を評価し，仮想ブラウザが通信量を増加させずに悪意のある Web ページを無害化できるという結論を得た（5 節）．

1.3 本論文の構成

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

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他の手法との併用を前提とする．これは次の理由による：

(1) 既存の手法と併用した方が，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．

既存の閲覧手法との併用の仕方には色々な選択肢があるが，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，既存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，信用できる Web サイトの一覧に追加し，次回からは既存のブラウザで直接アクセスさせる．

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（特にドライブ・バイ・ダウンロード攻撃）から閲覧者を保護することにある [*5]．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 ページへの防御能力を提供し，
(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．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・Local Storage・WebSQL などの永続化機能を使う Web アプリケーションをも利用できる

ことを意味する．

一方で，既存のブラウザを完全に置換することは仮想ブラウザの目的ではない．なぜならば，既存のブラウザは多くの機能を持っており，それらすべてを実装し仮想化することは，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，悪意のある Web ページを，

(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 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) 変換した画像を閲覧者のブラウザに表示させる

ことで無害化する [*6]．

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 の閲覧に関しては既存のブラウザと概ね同等である [*7]．これは，仮想ブラウザ

[*5] 本論文では言及しないが，本研究の目的には，RIA（Rich Internet Application）技術の非互換性によって発生する RIA の移植性の問題を解決することも含まれる．我々は中間表現とフレームワークとを用いる解決法 [24] を既に提案しているが，本研究はブラウザを差し替えることで別の角度からの解決法を提供する．

[*6] 特定の状況下 [11] では悪意のあるソフトウェアを画像に埋め込むことも不可能ではないが，仮想ブラウザがセキュリティを強化することは定性的に考えて明らかである．

[*7] 厳密には，隔離環境で動作させるブラウザの実装に依存する．

図 1　DPSWS2013 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 DPSWS2013 Web Page.

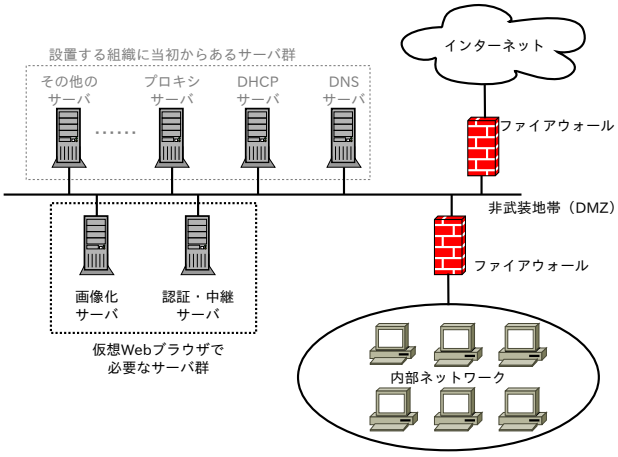

図 2　仮想 Web ブラウザのネットワーク構成
Fig. 2　Network Structure of Virtual Web Browser.

が，従来のブラウザが同一プロセス内で行っているイベントの発生と処理とに関する処理を，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 の処理能力に関しても本質的には同等と言える．

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，画像化に使う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，隔離環境で動作させるブラウザを変更して対処するしかない．一方で，多くの Web ページが将来的に HTML5 に移行することでこの問題は軽減されると我々は考える．

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，セキュリティの向上にもつながるため，利便性とセキュリティとのトレードオフの関係になる．

2.5　仮想 Web ブラウザのスクリーンショット

図 1 は，DPSWS2013 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．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挙げる．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 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」とから構成される．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を示す．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 上に配置し，
(3) 閲覧者の PC を内部ネットワーク上に配置する

必要がある．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（被害を完全に防ぐわけではない）[*8]．これは，一般に，内部ネットワーク・DMZ 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同様である）．

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混在して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 [10] を使

[*8] 厳密には，攻撃者は，当該サーバを完全に乗っ取ることで，仮想ブラウザの代わりに悪意のある仮想ブラウザを返すなどして，閲覧者を攻撃することができる．一方で，この手のリスクは通常の HTTP プロキシサーバなどでも同様であり，サーバのシステムファイルのチェックサムを定期的にチェックするなどの従来からの運用手法と組み合わせることで，リスクの低減が可能である．

用することを想定している．PAC ファイルを用いることで，サーバ側で一括して，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）

を指定することが可能になる [*9]．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可能になる．

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 [5] 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，画像化サーバのプロセスの隔離に使用される）；
(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 [*10]；
(4) 画像化サーバは，認証情報に基づいて画像化プロセスを隔離された環境で起動する；
(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 応答を隔離された環境で外見が等価な画像に変換する；
(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；

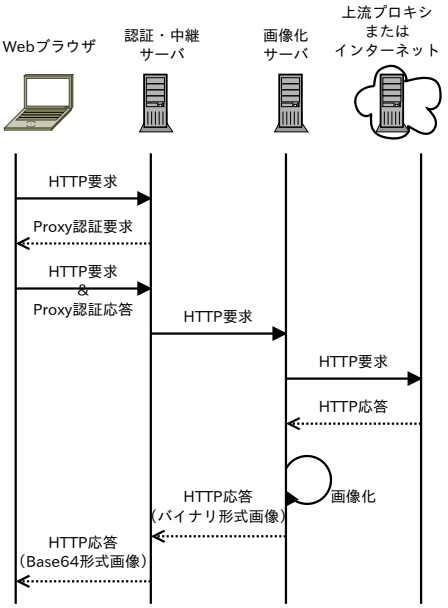

図 3 仮想 Web ブラウザの動作概要
Fig. 3 Behavioral Overview of Virtual Web Browser.

(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 [13] 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 `<img>` 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 Web 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が新しい 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 [*11]．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`）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とイベント情報を共有する．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 [*12]．

[*9] 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．
[*10] 一度認証が成功すると，ブラウザはその結果をキャッシュするため，閲覧者が再度の認証を要求されることはなくなる．
[*11] 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 を使用する場合と比較して効率が悪い．
[*12] ブラウザの画面サイズが変更されると，仮想ブラウザは新しい画面サイズを画像化サーバに通知する．これによって，画像化サーバは，常に正しい解像度で Web ページを画像化することができる．

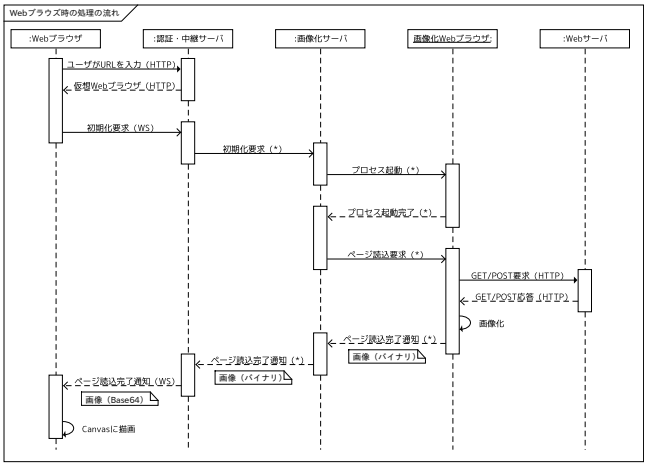

図 4　仮想 Web ブラウザのシーケンス図（Web ブラウジング時）
Fig. 4　Sequence Diagram of Virtual Web Browser (When Browsing Web).

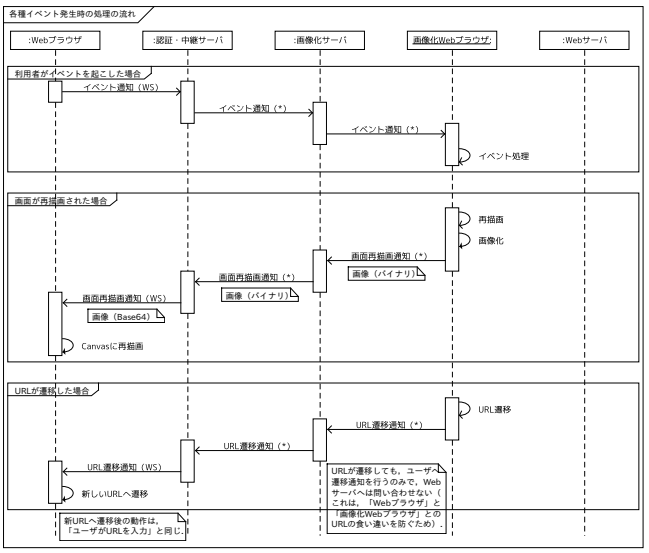

図 5　仮想 Web ブラウザのシーケンス図（イベント発生時）
Fig. 5　Sequence Diagram of Virtual Web Browser (When Events Fired).

3.3　プロセスの隔離

プロセスを隔離するためには仮想化技術（6 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，Unix の API を用いる方法を選択した．具体的には，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，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)` を用いる．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部が多くの Unix 環境で稼働することが期待できる．

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）攻撃とみなされ阻止されるためである [*13]．

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．これは，複数のウィンドウを 1 枚の画像に変換することが困難なためである．

[*13] 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などのデメリットの方が大きいため，HTTPS で動作させることを我々は想定していない．

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　Used Software for Virtual Web Browser Implementation.

ソフトウェア	バージョン	用途
jQuery	2.0.3	JavaScript ライブラリ
PhantomJS	1.9.1	画像レンダリング
Node.js	0.10.17	プロキシの実装
Apache HTTP Server	2.4.6	Web サーバ
CentOS	6.4	OS

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挙げる．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 [9] 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 [4] は JavaScript で命令を記述できる WebKit ベースの Headless[*14] なブラウザであり，画像化サーバを実装するために使用した．Apache HTTP Server（以下，Apache）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．Node.js [6] は，JavaScript で命令を記述できるサーバサイドのフレームワークであり，Apache の背後で PhantomJS とやり取りをする認証・中継サーバを実装するために使用した．

[*14] GUI を必要とせずに動作するソフトウェアのこと．GUI を必要とするソフトウェアと比較して，必要なリソースが少なく済むなどの利点がある．

```
<!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 -->
    <img id="mybrowser">
  </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).

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<img>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．

4.3　転送量の削減

我々は，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 [2] 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 Apache の標準モジュールでサポートされており，実装に必要なコストが低く済むためである．

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 Qt [1]）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．

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 Web Browsers

Web ブラウザ [1]	バージョン	gzip	compress	deflate
Internet Explorer	10.0	○	×	○
Firefox	22.0	○	×	○
Chrome	28.0	○	×	○
Opera	12.15	○	×	○

[1] Safari は，JVN（Japan Vulnerability Notes）が Windows 版の使用中止勧告 [8] をしているため評価の対象外とした．

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・GIF・JPG のみである．仮想ブラウザは標準で PNG を使用するが，必要に応じて変更することも可能である．ファイルサイズは一般に PNG>JPG>GIF となるが，PNG 以外を使用する場合には，JPG は非可逆圧縮であり GIF は表現できる色数に制限があること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，閲覧に専用のプラグインを必要とする Web ページ（例：Java Applet・PDF・Flash など）を画像化することはできない．一方で，専用のプラグインを動作させることは画像化プロセスが汚染されるリスクを高めることにつながるので，利便性とセキュリティとのトレードオフとなる．

5.　評価

我々は，仮想ブラウザの有用性を評価するために，仮想ブラウザと認証・中継サーバとの間の転送量の測定を行った．評価対象には Web 上のトラフィック量の上位 10 サイト（**表 3**）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をも含む）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**）がどれだけ増減するかを評価値とした．

評価の結果，仮想 Web ブラウザは表 3 の Web サイトの 90%を画像化することができた．また，表 4 から，画像を Base64 形式にするとサイズが増加するが，deflate で圧縮すると元の画像サイズとほぼ同じ大きさになることが分かる．

我々は，この評価の結果から，仮想ブラウザが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 [*15]．

6.　関連研究・関連技術

ここでは，本研究と関連する研究や技術を紹介し，本研究との差異について言及する．なお，多くの関連研究・技術が存在するため，本研究と近いもののみを取り上げる．

[*15] 厳密には，変換後の画像ファイルのサイズは，変換対象の Web ページの内容に大きく依存する．

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
Table 4　Image Sizes of Top Pages Shown in Table 3.

Web サイト [*1]	生サイズ [*2]	GIF	GIF Base64	GIF Base64 Deflate[*3]	JPEG	JPEG Base64	JPEG Base64 Deflate[*3]	PNG	PNG Base64	PNG Base64 Deflate[*3]
Facebook	427,250	30,571	41,301	30,474	61,899	83,618	44,051	88,921	120,125	86,883
Google	137,713	18,773	25,362	18,389	31,980	43,202	18,438	45,392	61,321	42,490
YouTube	746,831	102,683	138,714	104,402	180,734	244,151	146,309	411,112	555,365	412,024
Yahoo!	1,339,045	232,524	314,112	237,551	334,810	452,290	295,554	584,987	790,247	590,363
Amazon.com	1,372,108	367,484	496,428	375,405	402,804	544,139	376,220	1,195,046	1,614,362	1,214,274
Wikipedia	99,614	129,001	174,268	131,207	257,770	348,219	223,221	275,909	372,721	272,946
Windows Live	296,740	76,452	103,278	77,470	62,930	85,013	53,079	267,576	361,463	271,142
QQ.COM	1,460,273	415,434	561,201	424,595	768,416	1,038,037	721,419	2,052,965	2,773,305	2,087,799
Taobao.com	1,521,830	428,970	579,486	438,589	435,753	588,649	395,496	1,548,689	2,092,090	1,579,810

1　Baidu.com は，我々の環境では画像化できなかったため評価対象から除外した．
2　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．
3　Deflate の圧縮レベルには zlib のデフォルト値（6）を用いた．

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	Facebook	http://facebook.com/
2	Google	http://google.com/
3	YouTube	http://youtube.com/
4	Yahoo!	http://yahoo.com/
5	Amazon.com	http://amazon.com/
6	Baidu.com	http://baidu.com/
7	Wikipedia	http://wikipedia.org/
8	Windows Live	http://live.com/
9	QQ.COM	http://qq.com/
10	Taobao.com	http://taobao.com/

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しない低コストで安全な 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」[17] 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 [*16]．

6.3　先行研究・類似研究

Palanques ら [12] 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 ページを隔離された環境で画像化し閲覧者を悪意のあるソフトウェア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ソフトウェアから保護することを前提としており，HTML5 準拠のブラウザのみで動作する．

Grier ら [3] 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 ページを隔離され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 とを必要とする点で異なる．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．

Wang ら [16] 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 Unix の API のみを用いて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，安全な Web ブラウジングを実現する手法と

[*16] 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．

して仮想ブラウザを提案した．評価の結果は，提案手法が，通信量を増加させることなく悪意のある Web ページを無害化できることを示した．この結果から，我々は，クライアント PC のセキュリティの強化に提案手法が有用であり，悪意のある Web ページを閲覧する際の 1 つの解決策たりえるという結論を得た．

今後の課題としては追加の評価を検討している．具体的には，画像化によるレイテンシの増大やバックエンドのスケーラビリティに関する評価，Web ページが表示されるまでの待ち時間などを含む利便性の評価などが挙げられる．

今後の展望として，我々は，仮想ブラウザの利便性・機能性の向上を検討している．一例として，Web ページの部分的な画像化がある．これにより，Web ページ全体を画像化した場合にリンク・テキスト・アニメーションなどの情報が失われてしまうことを防ぐことができる．他の例として，`<canvas>`要素の使用がある．これにより，Web ページ中のリンクの視認性の向上や，テキストの選択およびコピー，アニメーションの動作などが可能になると期待できる．

参考文献

[1] Digia Plc: Qt Project, Digia Plc (online), available from ⟨http://qt-project.org/⟩ (accessed 2013-09-01).

[2] 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 2013-06-28).

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

[4] Hidayat, A.: PhantomJS: Headless WebKit with JavaScript API, PhantomJS.org (online), available from ⟨http://phantomjs.org/⟩ (accessed 2013-08-19).

[5] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart: HTTP Authentication: Basic and Digest Access Authentication, The Internet Society (online), available from ⟨http://tools.ietf.org/html/rfc2617⟩ (accessed 2013-09-01).

[6] Joyent, Inc.: node.js, Joyent, Inc. (online), available from ⟨http://nodejs.org/⟩ (accessed 2013-08-19).

[7] JPCERT/CC: 踏み台にされる Web サイト—いわゆる Gumblar の攻撃手法の分析調—, JPCERT/CC（オンライン），入手先 ⟨http://www.jpcert.or.jp/research/2010/webdefacement-20101115.pdf⟩（参照 2013-07-01）.

[8] 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 2013-08-19）.

[9] jQuery Foundation: jQuery: The Write Less, Do More, JavaScript Library, jQuery Foundation (online), available from ⟨http://jquery.com/⟩ (accessed 2013-09-01).

[10] Microsoft: Chapter 26 - Using Automatic Configuration, Automatic Proxy, and Automatic Detection, Microsoft (online), available from ⟨http://technet.microsoft.com/library/Dd361918⟩ (accessed 2013-09-01).

[11] 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 2013-08-19）.

[12] 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. of 31st IEEE Symposium on Reliable Distributed Systems (SRDS)*, Irvine, pp. 402–403 (2012).

[13] S. Josefsson: The Base16, Base32, and Base64 Data Encodings, The Internet Society (online), available from ⟨http://tools.ietf.org/html/rfc4648⟩ (accessed 2013-09-01).

[14] 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 2013-07-24）.

[15] W3C: HTML5, W3C (online), available from ⟨http://www.w3.org/TR/html5/⟩ (accessed 2013-06-28).

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

[17] 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 2013-06-25）.

[18] 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）.

[19] 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 2013-07-01）.

[20] 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 2013-06-28）.

[21] 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 1 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/2010/02outline.html⟩（参照 2013-07-01）.

[22] 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 2013-07-01）.

[23] 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）.

[24] 早川智一，長谷川慎哉，吉賀祥太，疋田輝雄：中間表現とフレームワークを用いた Web アプリケーションのメンテナンス法の提案と評価，情報処理学会論文誌，Vol. 53, No. 11, pp. 2571–2583 (2012).

[25] 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 2013-07-24）.