ユーザトラッキングを防ぐための 連動して動作するWebブラウザのFirefoxにおける実装

境 柊亮[1,a)]　新城 靖[1,b)]

概要：現在，多くの人々がインターネットを利用しており，様々なWebページを閲覧している．その一方で，Webサイト上ではターゲット広告などを目的としてユーザトラッキングが行われており人々のプライバシが侵害されている可能性がある．そこで，本研究ではユーザトラッキングを防止するために連動して動作する2つのWebブラウザを利用することを提案する．本研究の方式では親ブラウザと子ブラウザという2つのブラウザを利用する．このブラウザでは親ブラウザで行われた操作が子ブラウザでも行われる．これによりそれぞれのブラウザからリクエストが送信される．本研究では，それぞれのブラウザから発生するリクエストを比較し，差を削除もしくはランダムなデータに置き換えることにより，リクエスト中からユーザトラッキングで使われる識別子や属性を削除する．また，本研究ではこの提案手法に基づきWebブラウザFirefox，および，Tor Browserを元に実装を行った．そして，実装したブラウザを用いて実験を行いリクエスト中からユーザトラッキングで使われる識別子や属性が削除されていることを確認した．

1. はじめに

現在，インターネットが広く普及し，多くの人々が毎日様々なWebページを閲覧している．Webサイトは様々な手法を用いてユーザを追跡するユーザトラッキングを行うことによって情報を収集している．このユーザトラッキングによって集められた情報はターゲット広告などに利用される．しかし集めれた情報が実際にどのように利用されているかをユーザが確実に知る方法がなく，ユーザのプライバシが侵害されている可能性がある．また，Webサイトがユーザトラッキングを停止するためのフォームを提供していることもあるが，実際にユーザトラッキングが停止されているかを確認する方法はない．ユーザトラッキングの手法としては，Cookieを利用するものが古くから使われている．近年では，それ以外にも，ブラウザの実行環境やブラウザに特有の値（Browser Fingerprint）を使用してユーザトラッキングを行うという手法も利用されている[1]．

本研究室では，このユーザトラッキングを検知するために双子のWebブラウザを提案した[2]．この双子のWebブラウザとは，**図1**のように親ブラウザと2つの子ブラウザによって構成されるブラウザである．このブラウザでは2つの子ブラウザにおいて親ブラウザ上で行われた操作と同一の操作が行われる．このブラウザでは，2つの子ブラウザから発生するHTTPリクエストを保存し比較することでユーザトラッキングで使われる識別子や属性を検出する．しかし，この研究では保存後に保存したリクエストの比較を行っているため，検出はできるがトラッキングの防止を行うことができなかった．また，この研究では2つの子ブラウザをほぼ同一の環境で動作させていたためFingerprintによるユーザトラッキングの検出を行うことは難しかった．

そこで本研究では，双子のWebブラウザの考え方を発展させ，ユーザトラッキングを防止することを目指す．この際，Webページから得られる情報を維持するものとし，レイアウトなどが崩れることは許容する．本研究では双子のWebブラウザと異なり，子ブラウザを1つだけ使用する．そして双子のWebブラウザと同様に親ブラウザで行われた操作が子ブラウザでも行われるようにする．操作を行った結果，HTTPリクエストが発生すると，リクエストがWebサーバに送信される前に，親ブラウザと子ブラウザから送信されるHTTPリクエストの比較を行い，差が存

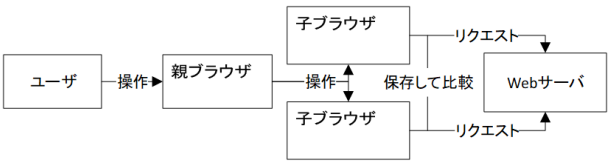

図1　双子のWebブラウザの構成

[1] 筑波大学
University of Tsukuba
[a)] shusukesakai@softlab.cs.tsukuba.ac.jp
[b)] yas@cs.tsukuba.ac.jp

```
POST /?id=webdglobd HTTP/1.1
Host: track.ss.gv.vc
Cookie: id=webdglobd; type=web
Content-Length: 19

resolution=1280x720
```

図 2 トラッキングを行うリクエストの例（親ブラウザが生成するリクエスト）

在した場合にはその差の削除もしくはランダム化を行う．これにより，ユーザトラッキングに使用される情報が送信されることを防ぎ，ユーザトラッキングを防止する．

2. ユーザトラッキングの手法

現在，利用されているユーザトラッキングの手法として Cookie を利用したものと Fingerprint を利用したものがある．Cookie とはブラウザ側に保存されるセッション管理になどに利用されるデータである．Cookie を利用したユーザトラッキングでは Cookie に識別子を保存しそれをもとにユーザを追跡する．

その一方で Fingerprint を利用したものでは，ブラウザを実行している環境やブラウザに特有の属性を利用する．これらの属性は単独ではユーザトラッキングを行うまでには至らないが複数の属性を組み合わせることで Fingerprint としユーザを追跡する．この属性の取得には JavaScript が利用される．Fingerprint として組み合わせる属性として，30 種類以上知られておりその例として利用可能なフォント，画面解像度，ユーザエージェント，キャンバスなどが存在する [3][4].

図 2 はブラウザがユーザトラッキングを行っているサーバに送信する単純化したリクエストの例である．この例では，1 行目の query string や cookie として id を送信している．この id を利用することによってトラッキングを行うことができる．また，このリクエストでは POST の body でユーザトラッキングでよく使われる属性の 1 つである画面解像度に関する情報を送信している．

3. 提案手法

本研究では連動する Web ブラウザを利用してユーザトラッキングの防止を実現する．本章ではその概要と構成について述べる．

3.1 概要

本研究で実装する，Web ブラウザの構成を**図 3** で示す．

親ブラウザと子ブラウザの 2 つのブラウザと，通信の比較などを行う親コントローラと，通信の中継や子ブラウザの操作を行う子コントローラによって構成される．

本研究において実装する Web ブラウザは以下の機能を

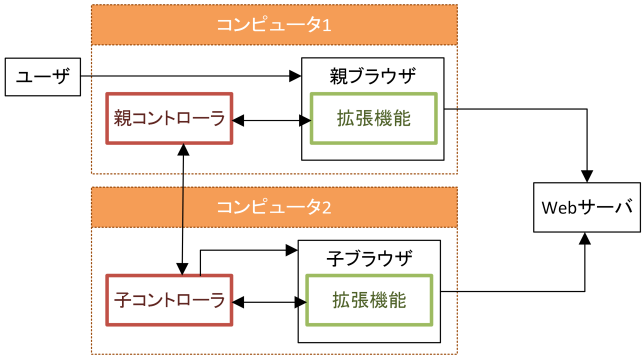

図 3 連動して動作する Web ブラウザの構成

```
POST /?id=gailamien HTTP/1.1
Host: track.ss.gv.vc
Cookie: id=gailamien; type=web
Content-Length: 19

resolution=1279x719
```

図 4 子ブラウザが生成したリクエストの例

持つ．

- 親ブラウザで行われた操作の子ブラウザへの反映
- リクエスト中の差の削除もしくはランダムなデータへの置き換え

ブラウザを複数動作させた時，メッセージの違いはブラウザが送信するリクエストだけでなく，サーバが送信するレスポンスにも存在する．本研究では，削除や変更を行うものは，リクエストのみとする．その理由は，ブラウザからのユーザトラッキングに使われる情報の発信を止めさえすれば，ユーザトラッキングは防げるからである．また，リクエストの形式はテキストが主であり，リプライと比較して単純であり，解析しやすい．

親と子の各ブラウザで新たな資源を取得しようとしても，すぐには Web サーバにリクエストを送らせず，拡張機能によって取得を行いそれをコントローラへと通知を行う．そして，コントローラからのリクエスト修正の指示を待つ．通知を受け取った親コントローラでは各ブラウザから送信されてきたリクエスト中の以下の項目を比較する．

- Request URL の query string
- ヘッダ（Cookie と Content-Length は別に処理）
- ボディ（x-www-form-urlencoded と JSON）

ヘッダのうち Cookie と Content-Length を別で処理しているのは Cookie はユーザトラッキングによく用いられるためであり，Content-Length はボディの長さに合わせて設定する必要があるためである．それぞれについての処理は後述する．

親ブラウザと子ブラウザでユーザトラッキングを行っている Web サイトの同じページへアクセスしたとする．親ブラウザは，図 2 に示したようなリクエストを生成したと

```
POST /? HTTP/1.1
Host: track.ss.gv.vc
Cookie: type=web
Content-Length: 0
```

図 5 置き換えられた後サーバへ送信されるリクエスト

する．子ブラウザは，**図 4** に示したようなリクエストを生成したとする．そのような場合，本研究ではそのリクエストをそのまま送信させずに，それを解析する．

図 2 と図 4 の比較すると，次の点が異なる事がわかる．

- 1 行目の request-line の query string の id=webdglobd と id=gailamien
- header の Cookie:の id=webdglobd と id=gailamien
- POST の body にある resolution=1280x720 と resolution=1279x719

本研究では，このような差分を発見した場合，リクエストを修正する．**図 5** は，その修正した結果の例である．この例では，修正方法としては削除を行っている．図 2 と図 5，または，図 4 と図 5 を比較すると分かるように，request-line や Cookie:にあった id=の部分が削除されている．また，POST の body にあった resolution=の行も削除されている．それに伴い Content-Length:も修正されている．

3.2 差の処理方法について

リクエスト中に差が見つかった場合に次のように処理することが考えられる．

- ブロック（リクエスト全体の送信を行わない）
- ランダム化
- 削除
- 固定値

このうち本研究では削除とランダム化を実装する．ブロックを行った場合については，リクエストが行われないためユーザトラッキングを防ぐことができるがページが正しく表示されなくなる可能性が高いため次のような場合に限定してブロックを行う．

- エラーが発生した場合
- POST の body が対応していない形式で差が存在する場合

一方で，削除や固定値に変更する場合にはアクセスごとのフィンガープリントが毎回同じものとなる．このため，利用者数が多い場合には複数人のフィンガープリントが同一となりユーザトラッキングを防止できるが利用者数が少ない場合にはトラッキングが行われる．ランダム化を行った場合にはアクセスごとに毎回フィンガープリントが異なることとなりユーザトラッキングを防止することができる．

4. 実装

提案手法を Web ブラウザ Firefox[*1]と，その Firefox をもとに改造されたブラウザである Tor Browser[*2]で実装した．本章では本研究で作成した Web ブラウザの実装について述べる．

4.1 Web ブラウザの一般的な動作

ウェブブラウザで HTML（Hyper Text Markup Language）をロードすると，ブラウザはその内容を読み取り DOM（Document Object Model）木に変換する．この初めに読み込む HTML を main frame と呼ぶ．JavaScript で書かれたコードはこの DOM 木に対して操作を行うことで表示内容を変更することができる．また，この DOM 木の DOM ノードにイベントリスナーを登録することによってその DOM ノードで行われたイベントを取得することができる．このイベントには様々なものが存在し，その例として，クリックやキー入力などのユーザの操作や，ページのロードの完了などが存在する．また，ブラウザは main frame の内容を元に JavaScript や Cascading Style Sheets（CSS），画像をロードする．さらにロードされた JavaScript からリクエストが発生することもある．このようなリクエストはユーザトラッキングで利用されることが多い．

Web サイトを閲覧したときに Web ブラウザ上で動作する JavaScript には主に 2 種類あり，1 つ目はサーバから送信されるもの，2 つ目は拡張機能によって追加されるものである．本研究では拡張機能によって様々な機能を実現する．

4.2 親ブラウザで行われた操作の子ブラウザへの反映

親ブラウザでは，行われた操作を取得するために，Web ページが読み込まれた際にスクリプトを挿入することによって DOM ノードにイベントリスナーを登録する．**図 6** に，挿入されるスクリプトの例を示す．このスクリプトは，任意ノードでクリックが行われると，sendSockWithDeleteText 関数が呼ばれるように設定している．この関数では，親コントローラに指示を送信することで親コントローラを経由して子コントローラに指示を行う．

子コントローラでは，親ブラウザでなされた操作を，Selenium[*3]というブラウザ自動操作フレームワークを利用して子ブラウザ上で行う．対応する操作は以下のとおりである．

- クリック
- キーボードからの入力
- ウィンドウのリサイズ

[*1] https://www.mozilla.org/en-US/firefox/
[*2] https://www.torproject.org/
[*3] https://www.selenium.dev/

```
window.addEventListener("click", sendSockWithDeleteText, true);
```

図 6 イベントハンドラへの登録と送信 (クリック)

基本的に親ブラウザで行われた操作と同じ操作を子ブラウザで行うが，ウィンドウのリサイズのみ，親ブラウザのウィンドウのサイズから 1px 引いたサイズにリサイズを行う．これにより，親子で差が生まれるようにし差の削除やランダム化を可能にしウィンドウサイズを利用したユーザトラッキングを防ぐことができる．

図 7 に，子コントローラにおいて Selenium の機能で操作を再現しているコードの主要部分を示す．この関数では親ブラウザから受け取った指示を元にノードを特定しクリックの操作を行っている．また，起動時のブラウザへの拡張機能の追加も同様に Selenium を利用して行っている．たとえば，今回の場合のように親ブラウザ上で行われた操作を取得するために拡張機能を利用したいことがある．そのような場合，本研究では，Selenium の機能を使って起動時にその拡張機能を自動的に追加する．

4.3 ログへの出力

コントローラは，各ブラウザにリクエストの修正を指示するのみでなくデバッグ，および，ユーザトラッキング手法の研究のためリクエストの差ををを出力する機能がある．**図 8** にその出力の例を示す．この出力は，外部のプログラム wdiff[*4] を利用して得たものである．図 8 中の背景色が赤と緑になっている部分がリクエスト中で差が見つかった箇所である．

4.4 ブラウザとコントローラ間のメッセージ形式

各ブラウザとコントローラ間は TCP による通信を行っている．この TCP によって通信を行う際，送受信するメッセージの形式は以下のような形式とした．

(1) length（32bit）
　後に続くデータの長さを表す．
(2) data
　ASCII 文字列

本研究では主に data として JSON 形式 [5] を用いている．それぞれで利用される詳細な形式は後で述べる．

4.4.1 リクエスト情報メッセージ

各ブラウザからコントローラに送信するリクエスト情報の形式は以下のようになっている．この形式は onBeforeRequest[*5] に設定した関数に渡される引数に基づいている．

- listen:string
  “beforeRequest” 固定
- method:string
  リクエストのメソッド（GET や POST など）
- requestId:string
  リクエストの ID
- tabId:integer
  リクエストを行ったタブの ID
- type:string
  リクエストのリソースのタイプ（main_frame や image など）
- url:string
  リクエストの URL
- timeStamp:number
  リクエストの発生時刻
- requestHeaders:array
  リクエストヘッダ
  array の要素は object となっておりそれぞれが name:string と value:string を持っている．
- requestBody:object
  リクエストボディ（存在する場合のみ）
  - raw:array
    onBeforeRequest で引数として渡される requestBody の raw:array のそれぞれの bytes を文字列に変換したもの（存在する場合のみ）
  - formData:object
    onBeforeRequest で引数として渡される requestBody の formData のそれぞれのキーと値の URL エンコードを行ったもの（存在する場合のみ）
    `{"name1":["v1", "v2"],"name2":["v3"]}` のような形式となっている．

4.4.2 処理結果のメッセージ形式

コントローラ側からブラウザの拡張機能へリクエストの処理結果を送る形式は以下のようになっている．

- block:boolean
  リクエストをブロックするかどうか
- url:string
  リダイレクト時の URL
- header:object
  リクエストヘッダ
- requestBody:string
  リクエストボディ

4.5 リクエストの比較

親コントローラではリクエストの比較を行う．その比較

[*4] https://www.gnu.org/software/wdiff/
[*5] https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/webRequest/onBeforeRequest

```
private void mousedownByXpath(String xpath) {
        WebElement element = driver.findElement(By.xpath(xpath));
        element.click();
}
```

図 7　子コントローラによる子ブラウザの操作

```
languages=08256fad734de00821ee24e4fb67500b
colorDepth=24
deviceMemory=undefined
screenResolution=d38197369162c6dabaa31dcb36060817
hardwareConcurrency=2
timezone=UTC
screenResolution=e468fc84f01ea75d23d00dfd8ae384f4
hardwareConcurrency=8
timezone=Asia%2FTokyo
sessionStorage=true
localStorage=true
indexedDB=true
```

図 8　wdiff の例

方法について説明を行う．

4.5.1　Request URL の query string の書き換え

本研究では URL をパースしクエリ部の差の削除やランダムなデータへの置き換えを行っている．

親ブラウザ，および，子ブラウザのそれぞれからのリクエストのクエリ部を Query-A, Query-B とすると次のような規則で差の削除やランダムなデータへの置き換えを行う．

- Quert-A, Query-B の双方に同一の名前のものが存在する場合．
  それぞれの値を比較して同じであれば残し，異なっていれば名前と値のペアを削除するか，もしくは値をランダムなデータ（0 から 99 の整数）に置き換える．
- Quert-A, Query-B のどちらか片方にしか存在しない場合．
  その名前と値のペアを削除する．

4.5.2　ヘッダの差の書き換え

本研究ではヘッダの差の削除やランダムなデータへの置き換えも行っている．ヘッダの差の書き換えでは “Cookie” だけ特殊な扱いをしている．これは Cookie が key-value 形式となっているためである．Cookie 以外に key-value 形式のものがあり値が異なっていた場合にはそのヘッダがまるごと削除もしくはランダムなデータに置き換えられる．Cookie 以外のヘッダは以下の様な規則で差の削除やランダムなデータへの置き換えを行う．

- 双方に同一の名前のヘッダが存在する場合．
  それぞれの値を比較して同じであれば残し，異なっていればそのヘッダを削除するか，もしくは値をランダムなデータ（0 から 99 の整数）に置き換える．
- どちらか片方にしか存在しない場合．
  そのヘッダを削除する．

Cookie は “`name1=value1; name2=value2`” のような形式となっている．本研究ではヘッダ中の Cookie をパースし差の削除やランダムなデータへの置き換えを行っている．Cookie は以下のような規則で差の削除やランダムなデータへの置き換えを行う．

- 双方に同一の名前の Cookie が存在する場合．
  それぞれの値を比較して同じであれば残し，異なっていればその Cookie を削除するか，もしくは値をランダムなデータ（0 から 99 の整数）に置き換える．
- どちらか片方にしか存在しない場合．
  その Cookie を削除する．

4.5.3　ボディの差の書き換え

本研究では以下の 2 つの形式を対象としてリクエストボディの差の削除やランダムなデータへの置き換えを行っている．対応していない形式の場合には比較のみを行い，異なる場合にはリクエストをブロックするように各ブラウザに指示する．

- application/x-www-form-urlencoded
- application/json（JSON 形式）

また，先頭文字が “{” の場合には JSON 形式としてのパースを試みている．さらに “=” や “&” が含まれている場合には application/x-www-form-urlencoded として扱っている．

application/x-www-form-urlencoded は URL のクエリと形式が同様であり，差の削除やランダムなデータへの置き換えも同様に行っている． 4.4.1 項の formData が存在する場合にはその値を利用する．この場合には後述する JSON 形式の場合の規則を適用したあと，application/x-www-form-urlencoded の形式に変換する．

JSON 形式では値として object, array, string 等の値をとる [5]．object の場合は以下のような規則で差の削除やランダムなデータへの置き換えを行う．

- 双方に同一の名前の要素が存在する場合．
  - 要素の型が異なる場合
    その要素を削除する．
  - 要素の型が object の場合
    それぞれの値で再帰的に差分の削除を行う．
  - 要素の型が array の場合
    それぞれの値で以下で述べる array の場合の規則をもとに比較と書き換えを行う．

– それ以外の場合
値を比較し同じであれば残し，異なっていればその要素を削除またはランダムなデータ（文字列，整数の場合は 0 から 99 までの整数，小数の場合は 0 から 10 までの値）に置き換える．

- どちらか片方にしか存在しない場合．
その要素を削除する．

array の場合はそれぞれの先頭の要素から順に比較を行っていく．長さが異なっている場合には短いほうの長さに合わせる．それぞれの要素を比較し差の削除やランダムなデータへの置き換えを行う際の規則は object の双方に同一の名前の要素が存在する場合と同じである．

4.6 ブラウザ本体の機能追加・変更

親ブラウザ，子ブラウザの実装は主にブラウザの拡張機能を利用して行っている．しかし，拡張機能のみでは実現することができない部分が存在した．その拡張機能だけでは実現することができない部分についてはブラウザ本体への変更を行った．追加・変更を行った機能は以下の 2 つである．

- TCP で通信を行うための JavaScript API
- webRequest API の onBeforeRequest でのリクエストボディ，リクエストヘッダの変更機能

以下にそれぞれの説明を行う．

4.6.1 TCP 通信 API

ブラウザでは HTTP などで TCP を利用しているが，TCP を直接 JavaScript から利用する方法はない．そこで，Firefox，および，Tor Browser へ TCP で通信を行うための JavaScript API の追加を行った．これは 4.4 節で述べたコントローラと拡張機能間での通信を行うためである．このために “dom/base/” と “dom/webidl/” 以下にファイルを追加し，関連するファイルの変更を行った．追加した API は以下の 4 つである．

- connect
TCP で接続を行う．
本研究では localhost で動いてるコントローラにしか接続を行わないため接続先アドレスは固定とした．
- read
ソケットからデータを読む
データは 4.4 節で述べたメッセージ形式の data 部のみが取り出されて渡される．
- write
ソケットにデータを書き込む
データは 4.4 節で述べたメッセージ形式に自動的に変換が行われる．
- close
ソケットを閉じる

4.6.2 webRequest API の onBeforeRequest

webRequest API[*6] とはブラウザの拡張機能からリクエストの情報の取得やリクエストの編集を行うための API である．そのうちの onBeforeRequest ではリクエストボディの取得や，リクエストのキャンセルが可能である．この API は，拡張機能で URL を元にリクエストをブロックするために利用される．本研究では，リクエストボディの取得のみではなく変更を行う必要がある．しかし，この onBeforeRequest ではリクエストボディの内容取得は行えるが，リクエストボディの変更を行うことはできない．また，本研究ではリクエストヘッダの取得と変更を行う必要があるがこれも onBeforeRequest では行うことができない．リクエストヘッダの取得と変更を行うためには別の API である onBeforeSendHeaders[*7] を利用する必要がある．そこで本研究では webRequest API の onBeforeRequest に対して以下の機能の追加を行う．

- リクエストボディの変更
- リクエストヘッダの取得と変更

webRequest API の onBeforeRequest はリスナー関数を登録することによって利用できる．このリスナー関数はリクエストの発生前に呼び出されることとなる．また，呼び出される際にリクエストの情報が引数として 1 つのオブジェクトにまとめられて渡される．この情報の中にはリクエストの URL などが存在し，例として URL の場合は引数を arg とすると arg.url で URL を取得することができる．また，リクエストを変更する場合はこのリスナー関数の返り値によって指定することとなる．例として返り値として返すオブジェクト中の cancel を true とするとそのリクエストはキャンセルされる．

本研究では onBeforeRequest でリクエストヘッダの取得を可能にするために引数として渡されるオブジェクトに requestHeaders を追加した．この形式は onBeforeSendHeaders で渡されるものと同様である．

また，返り値として返すオブジェクトで以下の 2 つを追加で指定可能とした．

- requestHeaders
リクエストヘッダの変更を可能にする．onBeforeSendHeaders と同様の形式となっている．
- requestBody
リクエストボディを編集可能とする．データの形式は ASCII 文字列となっている．

図 9 は，この拡張した API の使用例である．rewrite は実際に書き換えを行う関数であり，リクエスト中のリクエストボディを “`resolution=0x0`” に書き換え，リクエス

[*6] https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/webRequest
[*7] https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/webRequest/onBeforeSendHeaders

```
function rewrite(e) {
        return {requestBody:"resolution=0x0",
        requestHeaders:
                [{name:"Host",value:"track.ss.gv.vc"},{name:"Cookie",value:"type=web"}]};
}

browser.webRequest.onBeforeRequest.addListener(
        rewrite,
        {urls: ["http://track.ss.gv.vc/"]},
        ["blocking","requestBody","requestHeaders"]
);
```

図 9 拡張した API の使用例（onBeforeRequest）

トヘッダに “Host”, “Cookie” というヘッダを設定し，それぞれ値を “track.ss.gv.vc”, “type=web” にしている．addListener によって http://track.ss.gv.vc へのリクエストを rewrite 関数で処理を行うように設定を行っている．

5. 実験

4 章で述べた通り，本研究では Firefox と Tor Browser を元に実装を行った．

5.1 FingerprintJS

これらを 2 台のコンピュータで実行し実験を行った．コンピュータ 1 では Linux Ubuntu 20.04LTS を利用し，Tor Browser をもとに実装したブラウザを親ブラウザとして実行した．コンピュータ 2 はコンピュータ 1 上で Oracle VM Virtual Box[*8]を利用した仮想環境であり，OS として Windows 10 を利用し Firefox をもとにしたブラウザを子ブラウザとして実行した．また，Fingerprint によってユーザトラッキングを行うライブラリである FingerprintJS[*9]を利用して Fingerprint の属性を送信する Web サイトを作成した．本研究では実験として，このサイトに対し実装した Web ブラウザでアクセスを行い Fingerprint に利用される属性のうちどの属性が削除されるのか確認を行った．

表 1 に実験結果を示す．この表は FingerprintJS で取得を試みる 32 種類の属性を，次の 3 つに分類したものである．

- FingerprintJS で属性を取得できなかったもの
- 本研究において削除されるもの
- 削除されないもの

表を見ると 10 個の属性が削除されていることが確認できる．削除されないもののうち colorDepth，sessionStorage，localStorage，indexedDB，cookiesEnabled，monochrome，reducedMotion については多くの環境で同一の値になる．

表 1 FingerprintJS を試みる属性への対応

取得できない	削除	削除されない
domBlockers, deviceMemory, cpuClass, colorGamut, invertedColors, forcedColors, contrast, hdr	fonts, fontPreferences, audio, screenFrame, osCpu, screen Resolutions, hardware Concurrency, timezone, platform, canvas	languages, colorDepth, sessionStorage, localStorage, indexedDB, openDatabase, plugins, touchSupport, vendor, vendorFlavors, cookiesEnabled, monochrome, reducedMotion, math

languages は変化させると表示に影響が出る可能性が考えられる．plugins，vendorFlavors については空の配列，vendor については空文字列であった．openDatabase については Firefox では実装されていないため false となっている．touchSupport については実験環境にタッチパネルが存在しなかったため環境によっては変化する可能性がある．math については CPU アーキテクチャ，ライブラリなどにより変化する可能性が考えられる．削除されない属性や属性を取得できないということからブラウザの種類が特定される可能性があるが，複数人で同じ結果になると考えられるため個人を追跡することは難しいのではないかと考えられる．また今回は双方とも Firefox 系統をもとにしたブラウザであったためこれを別のブラウザをもとにして実装するとより多くの情報を削除できるのではないかと考えられる．Tor Browser 単体の場合と比較すると追加で削除できたものは osCpu と platform のみであった．これは本研究では動作している環境が異なっているため OS に関する情報が削除されるためであると考えられる．

5.2 Google での検索

多くの人が利用しており，ユーザトラッキングが行われ

[*8] https://www.virtualbox.org/
[*9] https://fingerprintjs.com/

図 10　https://www.google.com/での表示

ていると考えられる `https://www.google.com/` へのアクセスを行った．以降の実験ではどちらのブラウザも Firefox をもとにしたブラウザを利用している．**図 10** は本研究で実装したブラウザで `https://www.google.com/` へアクセスを行った場合の表示結果であり，通常の場合とユーザーインタフェースが異なっているが，利用には問題がないことがわかった．ユーザインタフェースが異なった理由は，User-Agent を削除したことによる．

また，検索を行ったところ通常とユーザインタフェースは異なるが，同じ検索結果が表示された．このことから得られる情報は同じであったと言える．また，このアクセスを行った際に発生する https://www.google.com/client_204 へのリクエストを記録し，通常の Firefox を用いてアクセスした場合の結果と比較を行った．client_204 へのリクエストを比較しているのは，最初のリクエストではブラウザ側にデータが保存されておらず，通常の場合とのリクエストの差が HTTP ヘッダの User-Agent に限られるためである．

送信されたリクエストを比較すると，文献 [3] で列挙された属性の中で HTTP ヘッダの User-Agent が削除されていることが確認された．また，文献 [3] で列挙された属性の中には存在しないが query string 中からユーザトラッキングでしばしば利用される，ウィンドウサイズであると考えられる biw や bih が削除された．文献 [3] で列挙された属性のうち，削除されなかった属性としては HTTP ヘッダの Accept，Accept-Language，Accept-Encoding がある．Accept は親子のブラウザの種類を変更すると削除される．Accept-Language は，親子のブラウザで言語設定を変更すると削除できる．しかしそれで表示内容まで変化してしまう．

5.3　Tranco

本研究で実装したブラウザを用いて Tranco[*10]の上位 26

*10 https://tranco-list.eu/

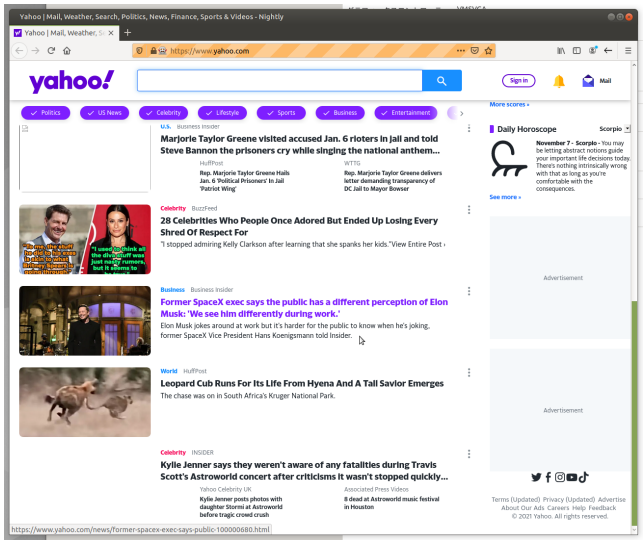

図 11　本研究で作成したブラウザを用いてアクセスした場合の https://www.yahoo.com/

個の Web サイト中のリダイレクトによる重複と，通常のブラウザで正常に表示できないものを除いた 23 個の Web サイトにアクセスを行った．その結果，問題なく表示できたものは 11 個，ページの上部のみが表示されるものが 4 個，その他，得られる情報が大きく異なるものが 8 個であった．

5.3.1　リクエストの差の許容

Tranco 実験の結果，下にスクロールするたびに JavaScript によってページの内容が追加されるようなページにおいて正しく動作しない場合があるということが判明した．これは追加の内容を JavaScript が取得する際に親と子のそれぞれのブラウザが次にどこを読み込むのかを示すための識別子を送信するが，この識別子がブラウザごとに異なり削除されてしまうのが原因である．

この問題への対応として，本研究では次のような機能を追加した．

- URL ごとに送信するリクエストの差を許容可能にする機能
  設定ファイルの一行に 1 つずつ許容する URL を示す正規表現を書くことにより，その正規表現に一致する URL のリクエストの差を許容することができる．
- 実行中にドメイン名と 4.4.1 項の type の組み合わせごとに許容するかユーザに確認する機能
  各組み合わせの最初のリクエストが送信されるたびにコンソールにメッセージが表示され許容するかどうかを `y` または `n` を入力することによって指定することができる．

5.3.2　https://www.yahoo.com/へのアクセス

下にスクロールするたびに JavaScript によってページの内容が追加されるようなページとして `https://www.yahoo.com/` が存在する．**図 11** は本研究で作成したウェブブラウザを用いてアクセスした場合の表示結果である．

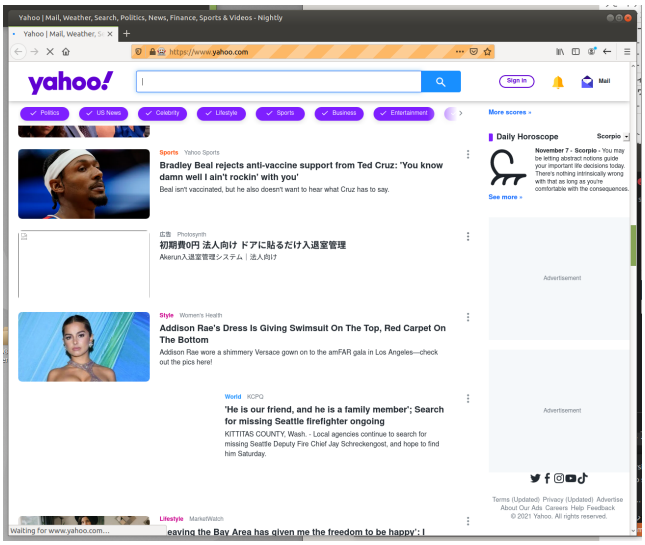

図 12　正規表現を用いて許可した場合の https://www.yahoo.com/

図 11 を見ると一番下までスクロールされていることがわかる．すなわち，本来長いページであるが，その途中までしか表示されなかったことがわかる．そこで 5.3.1 項で説明した機能を用いて，次のような正規表現で差を許容する実験を行った．

```
https://www\.yahoo\.com/tdv2_mtls_fp/remote\?ctrl=StreamGrid&.*
```

この時の表示結果が図 12 である．これを許容することにより，下にスクロールした際に追加で記事が表示されるようになることが確認できた．この機能を利用することによって，許可された URL に対してはすべてのユーザトラッキングに利用される属性や Cookie が送信される可能性がある．

5.4　課題

本研究で実装したブラウザでリクエスト数が多いページにアクセスした際にリクエストが完了するまで子ブラウザに操作が反映されないという問題がある．これは親ブラウザ上でリクエストの修正に関する処理を行い続けているため，親ブラウザ上でページ内で発生したイベントを取得できないことが原因である．

6.　関連研究

Datta 等の研究では Fingerprint によるユーザトラッキング防止ツールを以下の 3 種類に分けている [3]．

- 属性の標準化
  属性を標準化することによって複数のユーザが同じ Fingerprint となるようにする
- 属性の変動
  属性を変化させることによって Fingerprint を変化させる
- ブロック
  リストを元にしてすでに知られているトラッカーのブロックを行う

Blend In[*11] は一部の属性を変更することによって多く使われている OS で実行されているように見せかける．これは属性の標準化に分類されている．このようなものでは Fingerprint に利用される属性が増えるごとに対応する必要があるが，本研究ではそのような必要はない．

PriVaricator は Browser Fingerprint を利用したユーザトラッキングを防止するために，Fingerprint に使われる属性の値を変化させる [6]．FP-Block では Fingerprinting に利用される属性値のセットを Web サイトごとに切り替えることによって，Fingerprint を利用したユーザトラッキングを防止する [7]．これらの研究は属性の変動に分類される．これらの研究でも同様に Fingerprint に利用される属性が増えるごとに対応する必要があるが，本研究ではそのような必要はない．

Disconnect[*12] は様々なブラウザで利用することのできるユーザトラッキングを防止する拡張機能である．これはブロックに分類される．Disconnect ではリストを使用しているため，リストに存在しないものについてはユーザトラッキングの防止を行うことができないという問題点が存在する．本研究ではリストを使用しないため，このような問題は発生しない．

本研究では利用するブラウザの一種として Tor Browser を利用している．Tor Browser ではユーザトラッキングを防止するために，もととなった Firefox から Fingerprint に使われる属性の値を変化させている．そのため本研究で Tor Browser をどちらかのブラウザとして使うことによって，Tor Browser で変化させている属性についても削除されることとなる．その一方で，本研究では正しく表示することができないページでは一部のリクエストを許可することにより表示をしているが，この場合には Fingerprint に利用される属性が送信されてしまう．

7.　まとめ

本研究では，得られる情報を維持したままユーザトラッキングを防止することを目的として，親ブラウザと子ブラウザという連動する 2 つの Web ブラウザを利用したユーザトラッキングの防止を提案し，それを実装する．現在までに各コントローラと Firefox，および Tor Browser をもとにした各ブラウザの実装を完了し実験を行った．そして，追跡に利用される属性が削除されているとことなどを確認した．今後，異なる種類のブラウザで提案手法を実装し，評価を行う．

[*11] https://addons.mozilla.org/en-US/firefox/addon/blend-in/
[*12] https://disconnect.me/disconnect

参考文献

[1] Eckersley, P.: How Unique is Your Web Browser?, *Proceedings of the 10th International Conference on Privacy Enhancing Technologies*, PETS'10, Berlin, Heidelberg, Springer-Verlag, p. 1–18 (2010).
[2] 張　世申，新城　靖，三村賢次郎：個人情報及び個人識別子を含むファイルと通信を検出するための双子の環境，情報処理学会第 30 回コンピュータシステム・シンポジウム (ComSys2018)，pp. 29–36 (2018).
[3] Datta, A., Lu, J. and Tschantz, M. C.: Evaluating Anti-Fingerprinting Privacy Enhancing Technologies, *The World Wide Web Conference*, WWW '19, p. 351–362 (online), DOI: 10.1145/3308558.3313703 (2019).
[4] Mowery, K. and Shacham, H.: Pixel Perfect: Fingerprinting Canvas in HTML5, *Proceedings of W2SP 2012* (Fredrikson, M., ed.), IEEE Computer Society (2012).
[5] Bray, T.: The JavaScript Object Notation (JSON) Data Interchange Format, RFC 8259 (2017).
[6] Nikiforakis, N., Joosen, W. and Livshits, B.: PriVaricator: Deceiving Fingerprinters with Little White Lies, *the 24th International Conference on World Wide Web*, WWW '15, p. 820–830 (online), DOI: 10.1145/2736277.2741090 (2015).
[7] Torres, C. F., Jonker, H. L. and Mauw, S.: FP-Block: Usable Web Privacy by Controlling Browser Fingerprinting, *20th European Symposium on Research in Computer Security*, (online), DOI: 10.1007/978-3-319-24177-7_1 (2015).