利用者の端末の Fingerprint を CSS のみで採取する手法の提案と実装

武居 直樹† 磯 侑斗† 桐生 直輝† 塚本 耕司† 高須 航† 山田 智隆† 齋藤 孝道‡

†明治大学大学院, ‡明治大学
214-8571 神奈川県川崎市多摩区東三田 1-1-1
{ce46022, ce36004, ce36022, ce36032, ce46021, ce46032}@meiji.ac.jp, saito@cs.meiji.ac.jp

あらまし HTTPヘッダやJavaScriptなどから採取できるFingerprintを用いて利用者を識別する手法が普及しつつある. 一部の広告事業者はこの手法により利用者を追跡することから, 追跡を快く思わない利用者は対策ツールやJavaScriptの無効化などにより追跡を回避している. しかし, 我々の調査では, CSSを用いたFingerprintの採取は現状で対策が及んでいないことが分かった. そこで, 我々は, 本論文で, CSSを用いたFingerprintの採取法を提案する. この手法では, CSSのみでWebブラウザの判別や画面解像度, インストール済みフォントなどの取得が可能である. また, 本論文では提案した手法に対し既存の対策がどの程度有効であるのかを述べる.

Browser Fingerprinting only with CSS

Naoki TAKEI† Yuto ISO† Naoki KIRYU† Koji TSUKAMOTO† Ko TAKASU†
Tomotaka YAMADA† Takamichi SAITO‡

†Graduate School of Meiji University, ‡Meiji University
1-1-1, Higashimita, Tama-ku, Kawasaki-shi, Kanagawa 214-8571, JAPAN
{ce46022, ce36004, ce36022, ce36032, ce46021, ce46035}@meiji.ac.jp, saito@cs.meiji.ac.jp

Abstract It is getting popular that commercial Web sites collect a fingerprint to track Web visitor with HTTP header, JavaScript and so on. It is called as fingerprinting. Someone disables JavaScript or utilizes a prevention tool to avoid being tracked by them. However, we can say that countermeasures against fingerprinting only with CSS (Cascading Style Sheet) are not established. Therefore, in this paper, we propose a method of fingerprinting only with CSS. Moreover, we describe the effectiveness of our fingerprinting.

1 はじめに

Web 上での行動を追跡することで, 利用者の趣味嗜好に合わせた広告を出す, いわゆる, ターゲティング型広告という仕組みが利用されている. 利用者の Web 上の行動を追跡して記録することをトラッキングと呼び, その代表的な手法として, HTTP クッキーを用いる手法がある. HTTP クッキーを用いる手法では Web サーバが Web ブラウザ毎に一意な識別子を割り当てることで利用者を識別して追跡を行う. しかし, トラッキングされることを好まない利用者も存在し, Web ブラウザにより HTTP クッキーがブロックされるなど, トラッキングをさせないための技術が普及している.

他方, 近年, Peter Eckersley[1]により, HTTP クッキーを利用しない手法として, HTTP ヘッダや JavaScript から採取できる特徴を用いて利用者を識別する, いわゆる, Fingerprinting が提案された. これに対して,

採取される情報の書換え，制限，JavaScriptなどの実行の停止により，端末の識別を防ぐ技術が既に存在する[7][8][11].

そこで本論文では，端末の情報を採取する手法として，Cascading Style Sheet（以降，CSS という）のみにより端末の情報を採取する手法を提案する．また，提案手法に対しFingerprinting 対策ツールを用いた場合の影響を述べる.

Web Browser Fingerprinting に関する用語については，[2]に従い，それ以外は，[10]に従う.

2 関連技術

2.1 HTML レンダリングエンジン

HTML レンダリングエンジン（以降，レンダリングエンジンという）は，HTML を解釈し，文字や画像を Web ブラウザ上に描画するプログラムである．利用するレンダリングエンジンはWeb ブラウザにより異なり，挙動の差が存在する．また同じレンダリングエンジンにおいても，Web ブラウザのバージョンの違いにより挙動の差が存在する.

2.2 Cascading Style Sheet

Cascading Style Sheet （ CSS ）とは，HTML や XML の要素をどのように表示するかを定義する仕様であり，W3C により策定されている．現在までに CSS1，CSS2，CSS2.1 が勧告されており，また CSS3 についても仕様の一部が勧告されている.

2.2.1 Media Queries

Media Queries とは，CSS2 で定められた Media Types の拡張である．Media Types ではメディアの種類に応じた CSS の指定が可能であったが，Media Queries では更にメディア特性による指定が可能となった．メディア特性とは端末の状態を指し，代表的なメディア特性として画面の解像度や向きがある.

Media Queries が用いられている例として，Web ページにアクセスしてきた利用者のデバイスに合わせて自動的に Web ページのデザインやレイアウトを変えるレスポンシブ Web デザインが挙げられる.

2.2.2 @font-face

@font-face は CSS3 で定められた仕様である．Web ブラウザに表示する文字のフォントとして，Web 上からダウンロードさせたフォントファイルを適用させる，いわゆる，Web フォントと呼ばれる技術に利用される．利用者の端末に指定されたフォントが存在しない場合でも，Web ページの作成者が意図したフォントを，閲覧者の Web ブラウザに表示することが可能となっている．図 1 に@font-face の使用例を示す.

```
@font-face{
   font-family: 'foo';
   src : local("Meiryo"), url(“sample.ttf”); }
body{ font-family: 'foo'; }
```

図 1 @font-face の使用例

font-family では任意の名前を与え，src ではフォントのソースを指定する．src を複数指定した場合，レンダリングエンジンはソースが参照できるまで左から順に読み込む．参照できた場合，以降の記述は解釈されない．src には 2 つの指定方法があり，local()で指定した場合は利用者の端末内に指定されたフォントがあるかどうかを参照し，url()で指定した場合はその URI からフォントファイルをダウンロードしてフォントを適用する.

2.3 Fingerprinting 対策技術

Fingerprintingには，対策ツールが存在する．その例として，Mozilla Firefoxのアドオンである FireGloves[7]やNoScript[8]，通信を匿名化するWebブラウザであるTor Browser Bundle[9]などがある.

FireGlovesは採取されるFingerprintの書換えや，Webブラウザの機能を制限することによって，Webサーバが正確なFingerprintを採取す

ることを防ぐ. しかし, FireGlovesを利用すると画面解像度などの情報が書換えられ, Webブラウザ上での描画が崩れるなどのデメリットも生じる. また, Fingerprintを書換えた結果, 通常ではありえないFingerprintが生成されてしまうと, それ自体が一意な識別子として利用されてしまうこともある.

NoScript は, JavaScript や Java アプレットの実行, プラグインの起動に制限をかけることができる Web ブラウザの拡張機能である. NoScript は, JavaScript やプラグインの実行を禁止することで Fingerprinting を防ぐ.

Tor Browser Bundle は匿名で Web ブラウジングを行うことを目的とした Web ブラウザである. Tor Browser Bundle は Tor を利用するので, IP アドレスなど TCP/IP に関する情報を秘匿することが可能である.

3 CSS を用いた Fingerprinting

本論文で提案する CSS のみを用いた Fingerprinting について説明する.

提案手法では, Web サーバ上のコンテンツをリクエストするタイプの CSS のプロパティを利用する. コンテンツのリクエストには, Web ブラウザが適用したプロパティを Web サーバ側で判断するためのクエリ文字列を付与する. Web サーバ側では Web ブラウザから送られてきたリクエストのクエリ文字列を参照することで, Web ブラウザが解釈した CSS の記述を判断する. 図 2 にコードの例を示す.

```
p{background-image : url("database.php?
   property=background-image") ; }
```

図 2 CSS のプロパティの指定例

図 2 をレンダリングエンジンが解釈した場合, Web ブラウザから Web サーバ上のコンテンツである database.php のリクエストが Web サーバに送られる. リクエストのクエリ文字列の”property=background-image”から, 適用されたプロパティが background-image であると Web サーバ側で判断できる. database.php ではそれをデータベースに保存し, Web ブラウザに画像ファイルをレスポンスする.

3.1 CSS のプロパティへの対応状況の差による Web ブラウザ推定

レンダリングエンジン毎に解釈の差が生じる CSS のプロパティを用いて, Web ブラウザの推定を行う. レンダリングエンジンは, 非対応もしくは解釈が不可能な CSS の記述は無視するので, 各プロパティへの対応状況の差によって, Web ブラウザが送信するリクエストに違いが生じる. Web サーバでは受け取ったリクエストのクエリ文字列を参照することで, 利用者の Web ブラウザの CSS の実装状況を判断でき, Web ブラウザの種類やバージョンを推定できる. また, HTML の head 要素を CSS で装飾し, Web サーバのコンテンツをリクエストするプロパティを記述した場合, リクエストの送信の有無がレンダリングエンジン毎に異なる.

図 3 に CSS のプロパティを用いた Web ブラウザ推定のコードの例を示す. ここでは, test1～test7 までの 7 種のテストケースを示す.

```
div#mask-image{     /*test1*/
   -webkit-mask-image : url("database.php?
   property=maskimage") ; }
div#border-image{     /*test2*/
   border-image : url("database.php?
   property=borderimage") fill 10 / 10%/ 10px
   space ; }
div#image-source{     /*test3*/
   border-image-source : url("database.php?
   property=imagesource") ; }
div#box-sizing{     /*test4*/
   background : url("database.php?
   property=boxsizing") bottom right/ cover
   padding-box content-box ; }
div#gradient{     /*test5*/
   background : linear-gradient(to right,
   rgba(255,255,255,0), rgba(255,255,255,1)),
   url("database.php?property=gradient") ; }
div#multiple-bgs{     /*test6*/
   background-image: url("jpgfile/test.jpg"),
   url("database.php?property=multibgs") ; }
head{     /*test7*/
   background-image:url(“database.php?
   property=head”) ; }
```

図 3 CSS のプロパティを用いた Web ブラウザ推定のコード例

図 3 をレンダリングエンジンに解釈させた場合，対応しているプロパティのみが解釈され，Web サーバに database.php のリクエストが送信される．database.php は，データベースに Web ブラウザが適用した各プロパティ名を保存する．保存した情報から，Web ブラウザの対応状況を判断し，Web ブラウザを推定する．

3.2 Media Queries Fingerprinting

3.2.1 画面に関する情報の採取

画面解像度を指定した Media Queries を複数用意する．レンダリングエンジンは，端末の画面の幅が Media Queries で指定された幅の条件と一致する記述のみを適用する．よって，図 4 のコードで画面サイズが取得できる．

```
@media screen and (device-width: 1280px) {
   div#width{
      background-image : url("database.php?
      width=1280") ; }}     /*case1*/
@media screen and (device-width: 1366px) {
   div#width{
      background-image : url("database.php?
      width=1366 ") ; }}     /*case2*/
```

図 4 画面の幅の採取コード例（抜粋）

利用者の端末の画面の幅が”device-width”で指定した値と等しいとき，後ろに続けて記述されたスタイルが適用される．例えば，利用者の端末の画面の幅が 1366 ピクセルの場合，図 4 の case2 が端末の画面の幅と一致するので，case2 のみが適用され Web サーバへ”width=1366”をクエリ文字列に含んだリクエストが送信される．Web サーバはそのリクエストからアクセスした端末の画面の幅を取得できる．このメディア特性の値は 1 ピクセル単位で指定が可能であり 1 ピクセル毎に Media Queries を記述することで，端末の正確な画面の幅を採取できる．

その他に，表 1 の情報が取得できる．

表 1 Media Queries を用いて採取可能な情報

メディア特性	採取可能な情報
device-height:	画面の高さ
width:	Web ブラウザの幅
height:	Web ブラウザの高さ
orientation:	画面の向き （指定する値は landscape か portrait）
device-pixel-ratio	デバイスピクセル比

図 5 は Google Chrome 及び Opera 15 以上においてデバイスピクセル比を採取するためのコードの例である．Firefox から値を採取する場合はベンダプレフィックスとして”-webkit-“の代わりに”-moz-“を付ける必要がある．Opera バージョン 12 以前はベンダプレフィックスとして”-o-“を付け，指定する値は”3/2”のように，分数の形で指定する必要がある．IE はこのデバイスピクセル比用の Media Queries に対応していないため，値は採取できない．

```
@media screen and (-webkit-device-pixel-
ratio: 1){
   div#dpr{
      background-image : url("database.php?
      dpr=1"); }}
@media screen and (-webkit-device-pixel-
ratio: 1.5){
   div#dpr{
      background-image : url("database.php?
      dpr=1.5") ; }}
```

図 5 デバイスピクセル比の採取コード例（抜粋）

3.2.2 Mozilla 独自の Media Queries を用いた情報の採取

本節で挙げる Media Queries は Mozilla 独自の実装[4]であり，利用者が Firefox を用いていた場合のみ情報を採取することができる．メディア特性の値は”1”が指定されている場合は，そのメディア特性に端末が対応しているとスタイルが適用され，”0”が指定されている場合は，そのメディア特性に端末が対応していない場合にスタイルが適用される．図 6 に Firefox 独自実装の Media Queries を用いて，端末のメディア特性の対応状況を採取するコード例を示す．

```
@media screen and (-moz-touch-enabled: 1) {
   div#touch{
      background-image:
      url("database.php?touch=true") ; }}
@media screen and (-moz-windows-
compositor: 1) {
   div#dwm{
      background-image:
      url("database.php?dwm=true") ; }}
```

図 6 Firefox の Media Queries 指定例(抜粋)

Firefox の Media Queries により採取可能なメディア特性の情報を表 2 にまとめる.

表 2 Firefox の Media Queries を用いることで採取可能な情報

メディア特性	採取可能な情報
-moz-touch-enabled	タッチ画面への対応
-moz-windows-compositor	Desktop Window Manager の使用
-moz-windows-default-theme	Luna や Aero など Windows の既存のテーマの使用
-moz-windows-classic	クラシックモードの使用
-moz-mac-graphite-theme	Graphite の使用

Desktop Window Manager(以降, DWM という)とは Windows Vista, Windows 7, Windows 8 で使用されるデスクトップを描画するシステムである. Windows Vista 及び 7 では, デスクトップのテーマとして Aero テーマが使用されている場合にのみ DWM が使用され, Windows 8 では全てのテーマで使用される.

Graphite とは, Mac OS において, メニューやウィンドウなどのアピアランスに用いられる色の 1 つである. アピアランスには Graphite の他に Blue が存在する.

図 6 は全て値に"1"を指定しているため, 図 6 をレンダリングエンジンに解釈させた際, Web ブラウザは指定されたメディア特性に端末が対応している場合に, Web サーバへコンテンツのリクエストを送信する.

3.3 端末上のフォントの有無の判定

@font-face を利用して, 利用者の端末内のフォントの有無を判別できる. @font-face の src は local(), url()の順に記述し, local()に端末内の有無を確認するフォント名, url()には Web サーバのコンテンツを記述する. またコンテンツには有無を確認したいフォント名をクエリ文字列として付与する. 以下にフォントの有無を確認するコードの例を示す.

```
@font-face{
   font-family: 'font1';
   src: local('Arial'), url("database.php?
        fontname=Arial"); }
div#font1{ font-family: 'font1'; }
@font-face{
   font-family: 'font2';
   src: local('Century'), url("database.php?
   fontname=Century"); }
div#font2{ font-family: 'font2'; }
```

図 7 @font-face を用いた端末内のフォント有無判別コード例(抜粋)

図 7 のコードを解釈した際, local()に指定されたフォントが端末内に存在する場合, そのフォントが適用され, Web サーバへのリクエストは送信されない. 指定されたフォントが端末に存在しない場合, url()が解釈され Web サーバへ database.php を要求するリクエストが送られる. このリクエストのクエリ文字列としてフォント名を付与することにより, Web サーバは端末内に存在しないフォント名を採取できる. database.php は, フォントファイルをレスポンスし, クエリ文字列のフォント名をデータベースへ保存する. Web サーバは, CSS に記述したフォント名とデータベースに保存されたフォント名を比較することで, 利用者の端末内に存在するフォントを判断することが可能である.

4 実験

4.1 実験方法

CSS のみを用いて情報を採取する実験用の Web ページ(以降, CSS 実験ページという)を作成し, 3 章で述べたプロパティ, Media Queries や@font-face による情報の採取をした. @font-face の手法では, 3364 個のフォントを確認するコードを記述した. 3364 個のフォントについては, HTTP ヘッダ, Flash や JavaScript などから情報を採取する Web サイト(以降, 採取サイトという)[5]において, Flash により採取した 3364 個のフォント名を参考にした.

複数の端末の Web ブラウザから, 採取サイトと CSS 実験ページにアクセスし, 採取した値の比較を行った. また, フォントについては, 1 つの端末が 3364 個のフォントの有無の確認に要する時間も計測した.

実験を行った端末は以下の通りである.

- クライアント
  - ・OS:
    Microsoft Windows 7/8
  - ・ブラウザ:
    Google Chrome 36.0.1985.143
    Mozilla Firefox 30.0
    Internet Explorer 11
    Opera 12.14, 11.01
- Web サーバ
  - ・OS(distribution):
    CentOS 6.5 64bit
  - ・Web サーバ:
    Apache HTTP Server2.2.15
  - ・Web アプリケーションサーバ
    PHP5.3.3

4.2 実験結果

3.1 節の図 3 の 7 種のテストケースを用いて, 各 Web ブラウザの対応状況を調査した結果を表 3 に示す. Web ブラウザがテストケースに対応し, Web サーバにリクエストを送信する場合を〇で, テストケースに未対応でリクエストが送信されない場合を×で表す.

表 3 各テストへの Web ブラウザの対応状況

テスト / Web ブラウザ	test1	test2	test3	test4	test5	test6	test7
Google Chrome	〇	〇	〇	〇	〇	〇	〇
Firefox (22~30)	×	×	〇	〇	〇	〇	×
Firefox (15~21)	×	×	〇	×	〇	〇	×
Firefox (14 以前)	×	×	×	×	〇	〇	×
IE11	×	〇	〇	〇	〇	〇	〇
IE10/	×	×	×	〇	〇	〇	〇
Opera (11.1~12.17)	×	×	×	〇	〇	〇	×
IE9	×	×	×	〇	×	〇	〇
Opera 11.0	×	×	×	〇	×	〇	×
IE8	×	×	×	×	×	×	〇
Opera 10.1	×	×	×	×	×	×	×

同じ端末上の, 各 Web ブラウザから CSS 実験ページにアクセスをした結果, 端末から送られてきたリクエストからそれぞれ正しく Web ブラウザが推定できた. また, 端末を変えてアクセスをした場合も正しく推定を行うことができた.

CSS 実験ページの Media Queries により採取した画面の情報は, 採取サイトの JavaScript などによる採取と等しい情報が採取できた. Firefox については, 独自のメディア特性への端末の対応状況も採取できた. また同じ端末で Web ブラウザを変えた場合でも, Media Queries により採取した端末の画面の情報は等しい値であった.

@font-face によるフォントの有無を調べる手法について, 各アクセスに対して Flash を用いた場合の採取結果と CSS による採取結果を比較すると, 結果が一致するケースと一致しないケースあった. また, Windows7 の Google Chrome を用いて, 提案手法によってフォント情報を採取した際, 平均で約 51.79035 秒の時間を要した.

5 考察

5.1 プロパティによる Web ブラウザ推定について

今回 7 種のテストケースを用いることで，11 通りに Web ブラウザを分類することができた．今回のテストケース以外にも，対応状況の異なるプロパティを用いることで，より多くのブラウザの推定が可能である考えられる．また，セレクタも Web ブラウザ毎に対応状況の異なることがあるため，セレクタを用いることでも精度を上げることが可能であると言える．

5.2 @font-face による採取手法について

フォントの採取で，@font-face による手法と Flash による手法とで結果が異なった原因として，提案手法で予め指定されたフォント以外のフォントが端末内にインストールされていたことが挙げられる．提案手法では，@font-face で指定しなかったフォントが利用者の端末内にインストールされていたとしても，そのフォントについては有無を調べることができない．またフォントにはフォント名と PostScript フォント名があり，Web ブラウザによってはフォントの PostScript フォント名を指定しないと端末内を参照できないケースがあったことが挙げられる．これらに対しては，指定するフォントや，PostScript 名による指定を追加することで，Flash での採取結果に近い結果を得られるようになると考えられる．

今回得られた計測時間について，提案手法では端末と Web サーバ間で大量にリクエストとレスポンスが発生することからネットワークの状態や端末の処理速度によって，@font-face による手法に要する時間は変化すると考えられる．また，端末のフォントを正確に採取するために @font-face で多くのフォントを指定した場合，端末と Web サーバ間のリクエストとレスポンスの数が増加し，処理時間に影響を与えることも考えられる．

5.3 他のフォント採取手法との比較

利用者の端末にインストールされているフォントの情報を採取する，標準的な手法として Flash を利用する手法と JavaScript を利用する手法が存在する．

Flash による手法は，ActionScript の TextField.getFontList()や Font.enumerateFonts()を用いて端末にインストールされているフォントのリストを取得し，それを Web サーバ側に送信することでフォントのリストを採取する．Flash による手法では，利用者の端末に Flash Player がインストールされていることが前提となる．また Web ブラウザがプライベートブラウジングモードであった場合に上記 2 つのメソッドからフォントのリストの採取が不可能となる．提案手法にはこのような制約は存在しない．

JavaScript を利用する手法として，文字列の幅と高さの違いから判断する手法がある[6]．同じ文字列を 2 つ用意し，Web ブラウザがデフォルトで適用するフォントと，利用者の端末にあるか調べたいフォントをそれぞれに適用し，その高さや幅を比較する．利用者の端末内に指定したフォントが存在する場合，フォントを適用した 2 つの文字列の高さや幅が異なることからフォントが存在することがわかる．指定したフォントが存在しない場合，Web ブラウザはデフォルトのフォントを適用するため，2 つの文字列は同じ高さと幅になることから端末内にフォントが存在しないことがわかる．JavaScript による手法と提案手法を比較し，どちらの手法もより正確なフォントのリストを採取するためには予め多くのフォントを指定しておく必要がある点は同様である．JavaScript が実行できない場合においても提案手法ではフォントのリストが採取できる点が勝っていると言える．

以上の 2 つの手法では，採取したフォントのリストの送信は 1 度だけであるので端末や Web サーバに負荷がかかることなく実行時間が短いことが提案手法と比較して優れている点として挙げられる．

5.4 対策技術への適用

Fingerprinting 対策を施したブラウザを用いて，提案手法による Fingerprint の採取を行った．

Google Chrome を用いて，JavaScript が有効の場合と無効の場合における，提案手法の結果を比較した．その結果，どちらも Web ブラウザを Google Chrome と推定することができた．また，Media Queries を用いた情報の採取についても，等しい値を採取することができた．Firefox，IE を用いた場合も同様に，Web ブラウザの分類，Media Queries を用いて採取した情報は，JavaScript が有効か無効かに関わらず，等しい値を採取することができた．よって JavaScript の無効化は，提案手法への対策としては意味を成さないことがわかる．

FireGloves を用いた場合と用いない場合の結果を比較して，Web ブラウザの推定及び Media Queries により採取した情報については等しい結果となったが，フォントについては採取が不可能であった．

NoScript をインストールした Firefox についても，同様にフォントの採取が不可能であった．これらの拡張機能では，1 つのページに適用できるフォント数の制限をかけることや，@font-face の使用を不可能にすることができるので，このような結果となったと言える．

Tor Browser Bundle を用いた場合，アドオンとして含まれる NoScript が有効な場合はフォントの採取が不可能であったが，無効の場合は提案手法により端末のフォントの採取が可能であった．Web ブラウザ推定の結果は Firefox となり，Media Queries や@font-face を用いた手法では端末の値が正しく採取できた．よって，既存の Fingerprinting の対策ツールは，CSS による Fingerprinting の完全な対策とはならないが，一部の情報の採取に制限をかけることが可能であることが言える．

6　まとめ

本論文では，CSS のみにより Web ブラウザ種類やバージョンの推定を行う手法や端末の情報を採取する手法を提案し，実際に，CSS のみで利用者の Fingerprint が採取可能であることを示した．また Fingerprinting への対策を施した場合の提案手法への影響を検証し，一部の情報は採取が可能であることを示した．

参考文献

[1] Peter Eckersley, 2010,
How Unique Is Your Web Browser?
https://panopticlick.eff.org/browseruniqueness.pdf

[2] 齋藤孝道，磯侑斗，桐生直輝，Web Browser Fingerprinting に関する技術的観点での一考察，2014 年暗号と情報セキュリティシンポジウム予稿集．

[3] Media Queries
W3C Recommendation, 19 June 2012,
http://www.w3.org/TR/css3-mediaqueries/

[4] メディアクエリ - Web developer guide | MDN
https://developer.mozilla.org/ja/docs/Web/GGuid/CSS/Media_queries

[5] http://www.saitolab.org/fingerprint/

[6] JavaScript/CSS Font Detector
http://www.lalit.org/lab/javascript-css-font-detect/

[7] Firegloves
http://fingerprint.pet-portal.eu/?menu=6

[8] NoScript | Firefox アドオン
https://addons.mozilla.jp/firefox/details/722

[9] https://www.torproject.org/projects/torbrowser.html.en

[10] 齋藤孝道，マスタリングTCP/IP 情報セキュリティ編，オーム社，2013

[11] Ghostery, http://www.ghostery.com/