# 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

張　世申[1]　新城　靖[1]　三村　賢次郎[1]

**概要**：インターネットユーザは普段ネットワークサービスを利用する時に，意図せず個人情報や個人を識別できる情報をサービス提供者に送信することがある．ユーザのプライバシを保護するために，ファイルやネットワーク通信からそのような情報を検出し除去したいという要求がある．しかし，現在のアプリケーションは複雑であり，それは容易なことではない．本研究では，双子の環境という仮想実行環境を実装し，それを使ってファイルや通信から個人情報および個人を識別できる情報を検出することを提案する．双子の環境とは，プログラムファイルやデータファイル等の内容がほとんど同じである，または，類似のような 2 つの仮想実行環境である．双子の環境で同一のプログラムをそれぞれ実行し，同一の入力を与える．そして，2 つのプログラムの動作上の相違点を検出する．本研究では，双子の環境で双子のブラウザを実行する．双子のブラウザとは，ホストで動作する親ブラウザの操作を，双子の環境で動作している子ブラウザで再現するものである．本研究では，双子の環境を Docker コンテナを用いて実装した．また，双子のブラウザを Firefox，および，Google Chrome を用いて実装した．実装した双子の環境，および，双子のブラウザを用いて，ファイルやネットワーク通信から個人を識別できる情報を検出することができた．

**キーワード**：ユーザトラッキング, コンテナ,Web ブラウザ

## 1. はじめに

PC で動作するアプリケーションは，Web ブラウザのように明示的に通信を行うものだけでなく，オフィスツールのように，ユーザが意図しない通信を行うものがある．Web ブラウザであっても，ユーザトラッキングのための暗黙的に通信を行うことがある．このような意図しない通信により住所・氏名等の個人情報，および cookie のようにしばしば個人と結び付けられる情報 (以下，個人識別子) が送信されることがある．

たとえば，Web ブラウザは，個人情報，および，個人識別子として訪問履歴や，フォームの内容，cookie などをファイルに保存する．ユーザのプライバシを保護するために，ファイルやネットワーク通信からそのような個人情報や個人識別子を検出し除去したいという要求がある．しかし，現在の Web ブラウザやオフィスツールは，非常に複雑であり，これらの識別子がどのファイルにどのような形式で保存されているかを調べることは容易ではない．なお，以下では，個人情報と記載した時にも，個人情報と個人識別子の両方を含むものとする．

[1] 筑波大学
University of Tsukuba

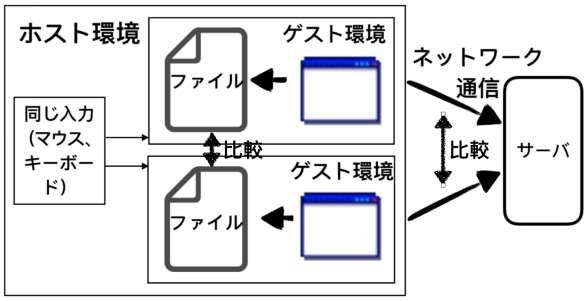


**図 1**　双子の環境

本研究では，双子の環境を実装して，個人情報および個人識別子が含まれているファイルや通信を検出する．双子の環境とは，プログラムファイルやデータファイル等の内容がほとんど同じであるような 2 つの仮想実行環境である (図 1)．人間における双子の研究では，異なる双子が異なる環境で育てられた際にどのような違いが生まれるかを調査する場合が多い．本研究で提案する双子の環境では，類似の 2 つの環境で同一のプログラムをそれぞれ実行し，同一，または，類似の入力を与える．そして，2 つのプログラムの動作上の相違点を検出する．

本研究では，双子の環境で双子のブラウザを実行する．双子のブラウザとは，ホストで動作する親ブラウザの操作

を，双子の環境で動作している子ブラウザで再現するものである．

本研究では，双子の環境を用いて内部で作動するアプリケーションが作成する全てのファイルの差分および発信するネットワーク通信の内容の差分を調査する．その差分に個人識別子が含まれる可能性が高い．さらにその差分を削除，または修正することで不用な個人識別子の発信を阻止することを試みる．

## 2. コンテナによる双子の環境の実装

本研究では，2 つの仮想実行環境を用意し，それぞれプログラムを実行し，仮想実行環境内のファイルを調査する．一般的な仮想マシンではホストはゲスト OS のファイルシステムを直接的アクセスできないという問題がある．そこで本研究では，コンテナという OS 層の仮想マシンを使う．

本研究ではコンテナを実装する仕組みとして Docker を使う． Docker では， Overlay File System というファイルシステムが利用可能である．このファイルシステムではゲスト OS のファイルをホスト OS からアクセスできるため，仮想実行環境内のファイルの差分を取得することが容易である．

ゲスト OS が生成したファイルは Overlay File System の Upper Layer に保存される．したがって，Upper directory を調査することで，書き込みがあったファイルを得ることができる．個人情報は，それらのファイルのどこかに保存されている．

### 2.1 ファイルの差分検出

図 2 に，Docker コンテナを用いたファイルの差分検出の仕組みを示す．まず，同じイメージを利用する 2 つのコンテナを起動し，対象となるプログラム (主に双子のブラウザ) を実行する. ホストからの入力を複製して，2 つのコンテナに与える．それからコンテナが生成したファイルをホスト OS で取得する．そして，その差分を調査し，個人情報を検出する．

Web ブラウザやその他のプログラムが個人情報を保存するためによく利用される形式としてはテキスト形式，データベース形式，および，マーシャリング形式の 3 つがある．マーシャリング形式としてはよく JSON と XML がよく使われる．本研究では Web ブラウザ Firefox と Google Chrome のファイル保存形式を調査した．その結果，利用されている形式には単純なテキスト，JavaScript，JSON，XML，SQLite，BerkeleyDB，LevelDB の 7 種類あることがわかった．

コンテナが出力するファイルがテキストファイルであれば，そのまま diff コマンドで差分を取ることができる．しかしながら，JSON や XML では，そのまま diff コマンドに与えても大量の出力がなされ，目的とする個人情報が埋もれてしまうという問題がある．また，diff コマンドは，データベース形式等のバイナリファイルを扱うことができない．

そこで本研究ではプログラムが生成したファイルをテキスト化し，さらに，前処理を行うことで，diff コマンドの入力として適した形に変換する (図 2)．まず File classifier で，ファイルを形式で分類する．次には，形式ごとにテキスト化する．主な形式について，テキスト化の方法を以下に示す．

- SQLite データベース．まず，sqlite3 コマンドの dump サブコマンドでテキストに変換する．次には，自動増加の主キー列を削除する．
- JSON ファイル．Python の json.tool ツールを利用して，JSON ファイルを標準化する．標準化とは，JSON ファイルに含まれる，Tab や Enter，Space を標準的なものに統一することである．
- JavaScript ファイル．Node.js のツール Javascript formatter により，標準化する．
- XML ファイル．xml-normalize を利用して，標準化する．
- BerkeleyDB ファイル．db_dump コマンドでテキストに変換する．
- LevelDB ファイル．LevelDB の C API により読み込み，テキストに出力するプログラムを本研究で作成した．

### 2.2 タイムスタンプの扱い

ネットワーク通信を行うプログラムは，様々なタイムスタンプをファイルに保存する．単純にファイルの差分を得ると，タイムスタンプの違いによるものが大量に生成され，重要な差分が埋もれてしまう．

本研究では，タイムスタンプを次の 2 つに分類して扱う．

・リモート: リモートの通信相手が指定したもの．例えば, HTTP の “Last-Modified:” ヘッダに由来するもの．

・ローカル: ローカルの OS からシステム・コールで取得したものに由来するもの．

リモート・タイムスタンプは，外部に発信されることが想定されている．例えば，HTTP の応答メッセージに含まれた “Last-Modified:” ヘッダの値は，同じコンテンツを再取得する時に，要求メッセージの “If-Modified-Since:” ヘッダに含まれて発信される．この値は，個人識別子としてユーザトラッキングに使われることがあることが知られている．一方，ローカル・タイムスタンプは，外部に発信されなければ，個人識別子にはなり得ない．したがって，双子の環境の実装では，ローカル・タイムスタンプの違いを排除したい．

本研究では，双子の環境で時刻関連のシステム・コール gettimeofday() と clock_gettime() をオーバーライドす

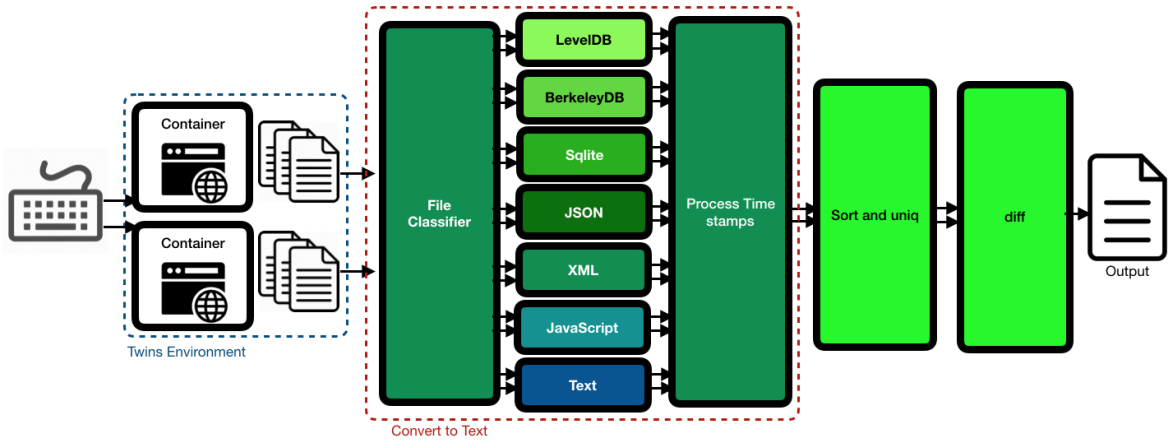


**図 2** Docker Overlay Filesystem によるファイル差分検出

ることで，ローカル・タイムスタンプの違いを排除する．そのため，本研究で作成した動的リンクライブラリを LD_PRELOAD で置き換える．置き換えたシステム・コールは，環境変数で指定された固定の時刻を返す．

このような処理を行ったとしても，アプリケーションの内部で独自にシステム・コールで得た時刻を加工して利用していることがある．例えば，Firefox では，システム・コール clock_gettime() で得られたナノ秒単位の時刻に，1 から 4 まで加えた値を利用している．この問題を解決するために，本研究では，テキスト化した後，ローカル・タイムスタンプとそれを加工したものと思われる数字を定数で置き換える．

### 2.3 ランダム性の排除

プログラムはマルチスレッドやネットワーク通信により非決定的な動作を行う．その結果，ファイルの内容に差が生まれ，本来検出したい個人情報を覆い隠してしまう．そこで，本研究では，そのようなランダムな行動を排除することを試みる．

まず，乱数によるプログラムのランダム行動を抑止するために，乱数デバイス/dev/urandom を置き換える．本研究では擬似乱数生成器デバイスドライバを作り，双子のコンテナのインスタンスに同じシードを与える．

マルチスレッドやマルチプロセスのアプリケーションのスケジュールも実行結果に影響を与える．たとえば，Chrome ブラウザの Task モデルは，ユーザ・インタフェースと入出力のスレッド以外にワーカスレッドが多数存在する．一つの作業がどのワーカスレッドにより実行されるか予想できない．その結果，Chrome ブラウザは，スレッド識別子のような予測できない結果を出力する．また，PC の負荷により動的にワーカスレッドの数を調整することもある [3]．そこで本研究では，テキスト化してダンプする時にスレッド識別子を含まないようにする．

### 2.4 ネットワーク通信の差分

双子の環境では，保存されるファイルの差分に加えて，そこから発せられるネットワーク通信の差分も調べる．そして，それに含まれる個人情報や個人識別子を検出し，さらに，ファイルに含まれる情報との突き合わせを行う．

双子の環境では，主に双子のブラウザを動作させる．それ以外にも，オフィスツール等のアプリケーションもネットワーク通信を行う．これらのアプリケーションがよく利用するプロトコルには次のようなものがある．

- HTTP，及び，HTTPS.
- WebSocket
- WebRTC(Web Real-Time Communication)

本研究では，まず，双子の環境の中で HTTP，および，HTTPS のクライアントを動作させ，それらが行うネットワーク通信の差分を調査する．HTTP，および，HTTPS は，オフィスツール等のアプリケーションが更新情報等を得るためにもよく使われる．

HTTPS は，TLS(Transport Layer Security) により通信路が暗号化されており，単純にパケットを保存しても内容を調査することはできない．そこで本研究は，MITM-Proxy(Man-In-The-Middle-Proxy) を使って TLS の復号を行い，通信内容を保存する．MITM-Proxy は，HTTPS だけでなく，HTTP も扱える．よって本研究では，HTTP，および，HTTPS の両方を統一的に MITM-Proxy で補足する．

MITM-Proxy は，Python API を提供する．本研究では，これを利用して，通信内容を補足する．図 3 に，MITM-Proxy を用いた通信内容の差分を取得する方法を示す．双子の環境では，双子のブラウザ，または，その他の通信を行うアプリケーションを実行する．これらのアプリケーショ

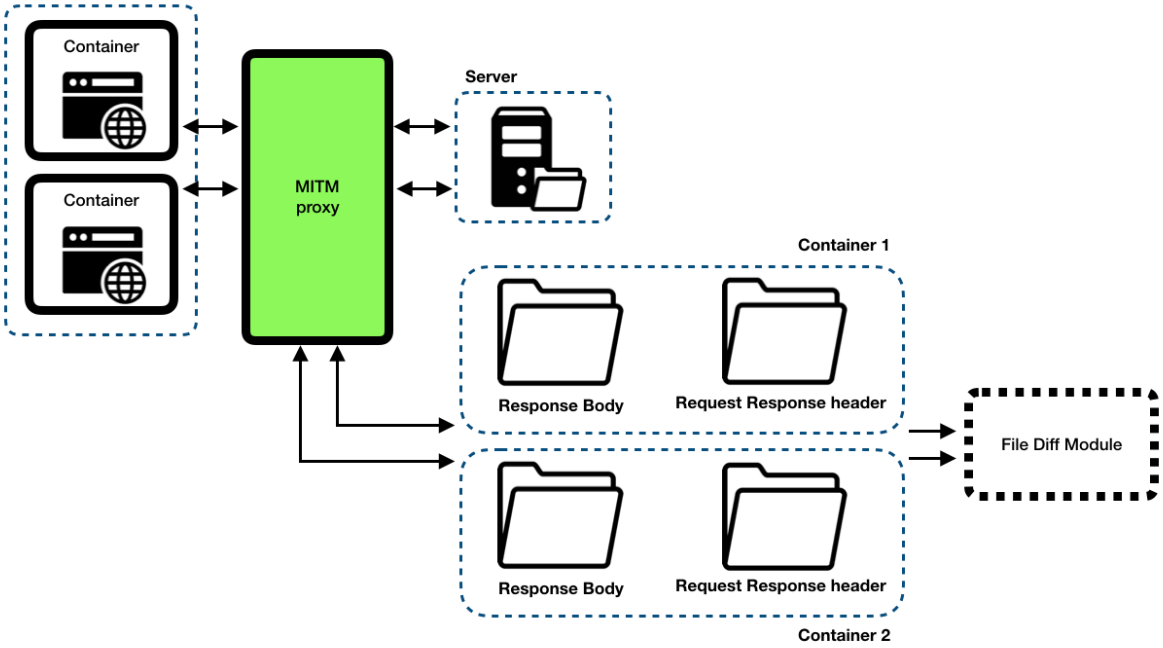


**図 3** MITM-proxy を用いた HTTP のメッセージの差分検出

ンでは，次のような設定を行う．

- インターネットへの通信を行う時には，プロキシとして，MITM-Proxy を経由する．
- TLS 証明書のルート認証局として，MITM-Proxy の内部の認証局を追加する．

そして，それ以外の通信を一切禁止する．その結果，双子の環境で動作するアプリケーションの通信は，MITM-Proxy に導かれる．

MITM-Proxy では，双子の環境で動作しているアプリケーションから接続要求が来ると，それを受け付け，そして本来のサーバに接続する．そして，HTTP の要求メッセージ，および，応答メッセージを中継する．HTTPS であれば，一度 TLS の暗号を復号し，メッセージを取得する．そして本来のサーバへ TLS で暗号化された通信路を開設して，メッセージを中継する．

MITM-Proxy は，次の 2 つのディレクトリにそれぞれファイルを作成する．

- 要求メッセージ，および，応答メッセージのヘッダ．
- 応答メッセージのボディ．

前者のディレクトリのファイルの内容は，次の項目から構成される．

- 要求メッセージの request line に含まれる query string
- 要求メッセージのオプションとなるヘッダ
- 要求メッセージのボディ
- 応答メッセージのヘッダ

後者のディレクトリのファイルの内容は，次の項目から構成される．

- 応答メッセージのボディ

各ディレクトリではファイル名として次の情報を用いる．

- HTTP の要求メッセージの request line．ただし，query string は削除する．
- 訪問の回数．そのサーバに要求を送った回数．

応答メッセージのボディを保存する時，次の HTTP のオプションを解釈する．

- `Content-Encoding: gzip`．圧縮されたデータを展開する．
- `Transfer-Encoding: chunked`．分割されたデータを結合する．

こうして保存したファイルは，2.1 節で述べたファイルの差分を取得するモジュールにより解析される．この時，応答メッセージの `Content-Type:`から得られた型を利用する．たとえば，HTML や JavaScript，CSS(Cascading Style Sheet) は，その型のテキストファイルとして扱う．イメージや 2.1 節で述べた形式以外の形式のファイルは，バイナリファイルとして扱う．

HTTP，および，HTTPS に加えて，MITM-Proxy は，WebSocket の通信を扱うこともできる．そこで本研究では，WebSocket の通信もファイルに保存する．この時，ファイル名は，HTTP と同じものに加えて，送信の方向を含める．ファイルの内容は，通信の内容である．

## 3. 双子のブラウザの実装

本研究では，Mozilla Firefox，および，Google Chrome を拡張して双子のブラウザを実装する．双子のブラウザは，図 4 のように親ブラウザと 2 つの子ブラウザから構成される．親ブラウザを操作するとそれが 2 つの子ブラウザ

でも再現される．

双子のブラウザで再現できる動作を以下に示す．

- ページを開く
- クリック
- フォームの input 要素の入力
- スクロール
- ウィンドウのリサイズ

双子のブラウザは，図 4 で示すように，親ブラウザの拡張機能，分配器，Selenium Standalone Server から構成される．親ブラウザの拡張機能は，WebExtensions[6] を用いて上記の操作を検知する．そのために，拡張機能は親ブラウザが表示する Web ページの DOM( Document Object Model ) ノードに対して予めイベントハンドラを登録する．イベントハンドラは，**操作情報**を分配器に送信する．操作情報は，次の情報を含む．

- 発生したイベント名．
- 対象の DOM ノード．DOM ノードを指し示す表現としては，XPath(XML Path Language) を用いる．
- テキストデータ．フォームの input 要素の入力の場合．
- イベントが発生したタブのタブ ID．

分配器と親ブラウザの拡張機能は，WebSocket でメッセージをやり取りしている．

分配器は，親ブラウザの拡張機能から操作情報を受けると，Selenium[7] を用いて子ブラウザを再現する．Selenium は，Web ブラウザの操作を自動化用するツールである．Selenium は，ブラウザに表示されている Web ページに JavaScript コードを挿入することができる．Selenium は，Java や Python, および JavaScript(Node.js) などの言語で記述されたプログラムから Web ブラウザを操作可能にする．本研究の分配器は，Python を使って実装した．また，Selenium を別のコンテナで利用するために，Selenium Standalone Server を子ブラウザが動作しているコンテナで立ち上げておく．

分配器は以下のようなメソッドを持つ．

- make_new_tab(url): 新しいタブを指定する URL で開く．このメソッドは，まず JavaScript コード ('window.open();') をブラウザに実行させ新しいタブを開き，Selenium の get メソッドで URL を開く．
- detect_tab(tab_id): タブ Id で指定するタブを操作可能な状態にする．タブの切替に Selenium の switch_to_window メソッドを利用する．
- clieck(xpath): 指定する要素をクリックする．Selenium の find_element メソッド，および，click メソッドを利用する．
- input(xpath, text): 指定する要素にテキストを書き込む．Selenium の find_element メソッド，および，send_keys メソッドを利用する．
- scroll(x, y): 指定する量だけスクロールする．JavaScript コード ('window.scrollTo(x, y)') をブラウザに実行させる．
- window_resize(x,y): Window の大きさを指定する大きさに変更する．Selenium の set_window_size メソッドを利用する．

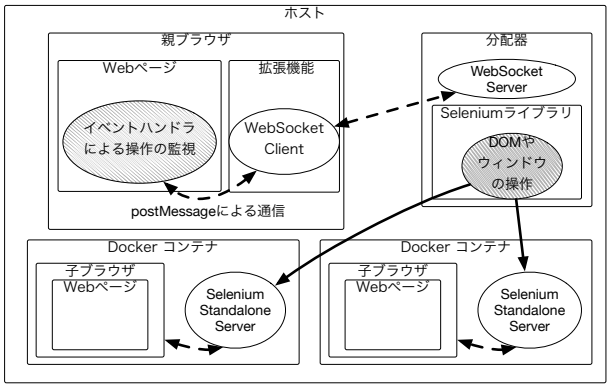


**図 4** Selenium による双子のブラウザの実装

親ブラウザでページを開くと，その URL とタブ Id が分配器に送られてくる．分配器は，親ブラウザのタブ Id と子ブラウザのタブ Id の対応を管理している．分配器はすでにタブが開かれていれば，detect_tab メソッドを用いて，そのタブを操作可能な状態にする．なければ，make_new_tab メソッドを使い新しくタブを開く．親ブラウザで，クリック， フォームの input 要素の入力，スクロール，ウィンドウのリサイズを行った場合にはそれぞれ対応するメソッドが呼ばれる．

## 4. 個人識別子を検出する実験

### 4.1 双子のブラウザが生成したファイルの差分

3 章で述べた双子のブラウザを用いてファイルや通信から個人識別子を検出する実験を行った. 実験の対象としては，ユーザトラッキングを行っている代表的な Web サイトとして Google の http://www.google.com/を選択した．これは，ログインを必要としない簡単なサイトである．

#### 4.1.1 Firefox

まず Firefox により実装した双子のブラウザをキャッシュサイズを 0MB に設定し，双子の環境で実行した．その結果の要約を表 1 に示す．コンテナで書き込みがあったファイルは，全部で 30 個であった．これらの 30 個のファイルに対して前処理を行って，テキスト化し diff コマンドで差分を取った結果，22 個のファイルには同一であることがわかった. 残りの 8 個のファイルでは，合計で 28 行の差があった．2.2 節，および，2.3 節で述べた方法に従って差分を削減した結果，最終的に差分は 4 つのファイルで 8 行になった．識別できないバイナリファイルはなかった．

表 2 に，発見した差分の一部を示す．cookies.sqlite には，HTTP cookies が保存されていることがわかる．places.sqlite は，訪問履歴を保持するファイルである．そ

**表 1** 双子の環境で書き込みがなされたファイル (Firefox)

| | 個人情報がない | 差分がある | |
|---|---|---|---|
| | | 不明なバイナリ | テキストと扱えるバイナリ (行数) |
| 更新されたファイル | 22 | 0 | 8(28) |
| 乱数生成器の置き換え | 26 | 0 | 4(10) |
| ローカルタイムスタンプの固定 | 26 | 0 | 4(8) |
| 前処理後 | 26 | 0 | 4(8) |

**表 2** 見つかったファイルの差分の例 (Firefox)

| ファイル名 | 環境 1 | 環境 2 |
|---|---|---|
| cookies.sqlite | value = 144=TiKON6Av0jIhBs_... | value = 144=ByW_1VAHxQkZtBU... |
| places.sqlite | guid = h1rV8kXSkMLh | guid = RZ4UHJH7cFa6 |
| sessionstore.js | “docshellUUID”: “{eec3c4c5-aee5-4d62-97ef-201625b37b9c}” | “docshellUUID”: “{0644435c-10b0-4e12-9d44-2aa871a4bfb1}” |
| prefs.js | user_pref(“browser.slowStartup.averageTime”, 16532) | user_pref(“browser.slowStartup.averageTime”, 14325) |

の中に，アクセスした URL が保存されている．このように，URL の中に，ユーザトラッキングに利用可能なタグが埋め込まれていることがわかる．

この実験の結果，提案手法により，ファイルを特定し，使用者の識別子を検出することができ，ユーザトラッキングに使われる個人識別子が含まれることが確認された．

#### 4.1.2 Chrome

次には Google Chrome により実装した双子のブラウザで 4.1.1 項と同じ実験を行った．結果の要約を表 3 に表す．差分があるファイルは Firefox より多かった．最終的には，8 つのファイルで，合計 12 行の差が見つかった．表 4 に，発見した差分の一部を示す．

現在のところ，次のような識別できない差分があるバイナリファイルが 3 個残された．

- Current Session
- Current Tabs
- Translate Ranker Model

### 4.2 ネットワーク通信内容の差分

Web ブラウザとして Firefox を用いてネットワーク通信の内容の差分を調査した．訪問したサイトは，4.1 節と同じである．結果の概要を表 5 に示す．通信の数は全部で 21 個あり，テキスト形式のファイル以外に PNG(Portable Network Graphics) 形式の画像が 6 個，ico(icon) 形式の画像が 1 個あった．20 個のメッセージの内容に差分があり，1 番目の通信だけは差分がなかった．

応答メッセージの内容に差分があるファイルが 2 個あった．それらの URL は www.google.com と www.google.com/gen_204 であった．また，URL が違う通信の数は 3 個あった．ネットワーク通信の差分はタイムスタンプと重複を除き 22 行あった．

表 6 に，見つかった差分の一部を示す．要求メッセージには，Cookies が現れている．これは，cookies.sqlite(表 2) にも含まれている．

## 5. 個人識別子の削除

2 章では，双子の環境を使って個人識別子を含むファイルとネットワーク通信を検出する方法について述べた．4 章では，実装した双子の環境でいつくかの個人識別子を見つけることができたことを示した．この章では検出した個人情報を削除するツールについて述べる．

### 5.1 ファイルから個人識別子の削除

現在の Web ブラウザやオフィスツールは，非常に複雑であり，多くのファイルをアクセスする．個人識別子は，これらのファイルのうち，更新されるものに保存される．本研究では，2.1 節の手法により，更新されるファイルを特定することができる．これらの更新されるファイルをすべて削除すれば，個人識別子も削除されるが利便性が大きく低下することがある．たとえば，Web ブラウザでは，ブックマークや個人の設定を保存するファイルが更新されるが，それらを全て削除すると，利便性が低下する．

そこで本研究では，更新されるファイルを，次の 3 種類に分類して考える．

- タイプ 1: アプリケーション自身が個人識別子を削除する機能を提供しているもの．たとえば，Web ブラウ

**表 3** 双子の環境で書き込みがなされたファイル (Google Chrome)

| | 個人情報がない | 差分がある | |
|---|---|---|---|
| | | 不明なバイナリ | テキストと扱えるバイナリ (行数) |
| 更新されたファイル | 98 | 3 | 47(114) |
| 乱数生成器の置き換え | 129 | 3 | 16(46) |
| ローカルタイムスタンプの固定 | 129 | 3 | 16(41) |
| 前処理後 | 140 | 3 | 5(12) |

**表 4** 見つかったファイルの差分の例 (Google Chrome)

| ファイル名 | 環境 1 | 環境 2 |
|---|---|---|
| Cookies | encrypted_value = v10D\6P\kC\4DB0... | encrypted_value =v10\80\A5\C380A... |
| Preferences | “http_original_content_length”: “5822543” | “http_original_content_length”: “5822508” |
| blob_storage | 0f4d89fa-3338-461a-8223-79fafb95b4ae | 4a59d2e5-f105-4ad4-995a-7186246adb0a |
| Local State | “variations_seed_signature”: “MEUCIQD31Hk3r25aa4Zigsjsy5s...” | “variations_seed_signature”: “MEUCIFj9DePjDdyFk...” |
| Network Persistent State | “network_stats”: “srtt”: 4890 | “network_stats”: “srtt”: 27660 |

ザなら，cookie を終了時に削除する機能がある．

- タイプ 2: ファイル全体を削除すると問題が生じるもの．
- タイプ 3: ファイル全体を削除しても問題が生じないもの．

本研究では，主にタイプ 2 のファイルについて，内部の個人識別子を特定し，削除するか，ランダムな値で置き換えるツールを作成する．

次のファイルを削除すると，利便性が低下することを確認した．

- Firefox places.sqlite．削除すると履歴機能が使えなくなる．
- Firefox prefs.js．削除するとアプリケーションの設定が失われ，初期状態にもどる．
- Firefox sessionstore.js．削除すると，ブラウザの「Restore Previous Session」の機能が使えなくなる．

本研究では，Web ブラウザの次のファイルから個人識別子情報を削除するツールを作成する．

- Firefox cookies.sqlite. persistant cookie の value をランダムな値で置き換える．
- Firefox prefs.js. 時刻に関する行 (browser.slowStartup.averageTime 等) を削除する．
- Firefox places.sqlite. guid をランダムな値で置き換える．
- Firefox sessionstore.js docshellUUID をランダムな値で置き換える．principalToInherit_base64 をランダムな値で置き換える．

### 5.2 ネットワーク通信から個人識別子の削除

ファイルからの個人識別子の削除は，プログラムを一度終了しなければ行うことができない．たとえば，Web ブラウザ Firefox の places.sqlite に含まれる guid は，5.1 節で述べたツールで削除できる．しかし，ブラウザの実行中には，このツールは動作しない．

アプリケーション自身が，個人識別子の送信を抑止する機能を持っていることがある．たとえば，Web ブラウザは，cookie を送信しない機能や，検索結果の URL から個人識別子を削除する拡張機能を持っている．Web ブラウザについては，このような既存の機能を利用すれば，かなりの個人識別子を削除することができる．しかしながら，このような Web ブラウザの機能には，ブラウザが行う暗黙的な通信を扱えないという限界がある．たとえば，Web ブラウザが SSL/TLS の証明書を検証するために Online Certificate Status Protocol(OCSP) に基づき行う通信を扱えない．また，Web ブラウザ以外のアプリケーションについては，ネットワーク通信から個人識別子を削除する機能はほとんど開発されていない．

本研究では，MITM-Proxy を用いてネットワーク通信から個人識別子を削除するしたいと考えている．

Web サーバは，IP アドレスやブラウザの Fingerprint 等のメタ情報を使ってユーザトラッキングを行うこともある．送信するメッセージの内容だけを操作しても，それには限界がある．本研究では，IP アドレスを隠すことについては，Tor[4] 等の既存の手法と組み合わせて用いることにする．

**表 5** 見つかったネットワーク通信の差分の数 (Firefox)

| | |
|---|---|
| 全部の通信数 | 21 |
| 差分がある通信 | 20 |
| 応答に差分がある数 | 2 |
| タイムスタンプと重複を排除の差分 | 22(行) |

**表 6** 見つかったネットワーク通信の差分の例 (Firefox)

| URL | 環境 1 | 環境 2 |
|---|---|---|
| www.google.com | Cookie: 1P_JAR=2018-10-30-10,NID=144=TiKON6Av0jIhBs_... | Cookie: 1P_JAR=2018-10-30-10,NID=144=ByW_1VAHxQkZtBU_... |
| www.google.com/gen_204 | ei: zTHYW-XZLYm78QXirYeIDA | ei: zTHYWBENxVnt4YGsqr4WR |
| ssl.gstatic.com/gb/images/i1_1967ca6a.png | age: 402757 | age: 402758 |

## 6. 関連研究

Qubes-OS[1] は高いセキュリティを実現するための OS である．Qubes-OS は仮想計算機モニタ (Xen) を用いて，アプリケーションを隔離された仮想実行環境で実行する．Qubes-OS には，複数の仮想実行環境のファイルを比較する機能はない．

Blink-Docker[2] はコンテナ中でブラウザを実行することで，Canvas Fingerprint を利用したユーザトラッキングからユーザを保護する．Canvas Fingerprint はハードウェアや OS のわずかな違いによってクライアントを特定する手法である．Blink-Docker はコンテナ技術を利用し，毎回 Fingerprint を変えることができる．本研究では，通信内容を検査し，そのようなユーザトラッキングを検出したいと考えている．

## 7. まとめ

本研究では個人情報および個人識別子を含むファイルと通信を検出することを目的として双子の環境を提案した．双子の環境では双子のブラウザを実行する．本研究では双子の環境を Docker コンテナで実装した．そして Overlay File System を用いてファイル差分を得て，それに含まれる個人識別子を検出した．また，Man-In-The-Middle Proxy を用いて，通信内容をファイルに保存，比較し，個人識別子を検出した．また，いくつかのファイル型について，個人識別子を削除するツールを作成した．

今後の課題は，通信内容から個人情報や個人識別子を削除するツールを実現することである．また，個人識別子を削除できるファイルの種類を増やしたいと考えている．

### 参考文献

[1] *Qubes OS - A reasonably secure operating system*: https://www.qubes-os.org/, Accessed: 2017-11-22.
[2] Pierre Laperdrix, Walter Rudametkin, Benoit Baudry: *Blink: A moving-target approach to fingerprint diversification*, 37th IEEE Symposium on Security and Privacy, Poster Session (2016), [Online] http://www.ieee-security.org/TC/SP2016/poster-abstracts/59-poster_abstract.pdf.
[3] Javier Verd, Juan Jos Costa, Alex Pajuelo: *Dynamic Web worker pool management for highly parallel javascript web applications*, Concurrency and Computation: Practice and Experience, 28(13), pp.3525-3539 (2015).
[4] Roger Dingledine, Nick Mathewson, Paul Syverson: *Tor: The Second-Generation Onion Router*, Proceedings of the 13th USENIX Security Symposium (2004).
[5] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, Claudia Diaz, *The Web Never Forgets: Persistent Tracking Mechanisms in the Wild*, Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, pp.674-689 (2014).
[6] *WebExtensions JavaScript APIs*: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API, Accessed: 2018-10-23.
[7] *Selenium - Web Browser Automation* : http://www.seleniumhq.org/, Accessed: 2018-10-23.