明治大学総合数理学部

2025 年度

卒　業　研　究

日本の主要 Web サービスにおける FIDO2/パスキー認証の導入実態と WebAuthn 設定の安全性評価

学位請求者　先端メディアサイエンス学科

田口 凱之

目次

第 1 章

はじめに

FIDO2 およびパスキーは，公開鍵暗号と認証プロトコル（生体認証や PIN 等）を組み合わせることで，フィッシング耐性を備えた次世代認証方式として普及が進んでいる．2023 年，Kuchhal ら [1] は，世界の主要 Web サービス (Tranco Top 1K) を対象に，FIDO2 の導入状況と RP（Relying Party）側のセキュリティ設定を調査し，多くのサービスで推奨設定が採用されておらず，開発用の仮想認証器であっても実サービスへ登録できてしまう例が多数存在することを報告している．しかし，彼らの調査は主にグローバルサービスを対象としており，日本の主要 Web サービスにおける FIDO2/パスキー導入状況や RP のセキュリティ設定についてはまだ明らかになっていない．

そこで，本研究では，日本の主要 Web サイトを対象に FIDO2/パスキー導入状況および RP のセキュリティ設定を調査する．Chrome 開発者ツールの仮想認証器機能を利用して秘密鍵クローン実験の PoC（Proof of Concept）を構築し，Kuchhal らの指摘する脆弱性を再現実験する．実験結果に基づき，仮想認証器を許容することのリスクを具体的に示す．

第 2 章

先行研究

2.1 Real-World FIDO2 Deployments の調査（Kuchhal ら，2023）

2023 年に Kuchhal らは，Tranco Top 1K の主要 Web サービスを対象に，FIDO2/WebAuthn の導入状況と RP 側の設定実態を調査した．同研究では，アカウント作成が可能な RP を対象に手動調査を行い，最終的に 29 件の RP を評価している．

また, RP のクライアント実装が呼び出す WebAuthn の登録 API（`navigator.credentials.create()`）[*]の引数をデバッガで観測し，RP が要求する認証器条件（例：認証器の種別制約，attestation [†]要求，ユーザー検証要求）を整理している．

さらに同一の RP 群に対して, Chrome DevTools の WebAuthn パネルが提供する仮想認証器[‡]を用い, protocol（CTAP2/U2F）[§]および transport（USB/BLE/NFC/internal）[¶]の組合せを変えながら登録可否を評価し，RP がテスト用の仮想認証器を正規の認証器として受け入れるかを測定している．

2.2 先行研究の知見と課題

先行研究は，緩和策として attestation に基づく識別・許可リストを挙げつつ，実測として attestation を要求する RP が 14/29 に留まり，仮想認証器を登録可能な RP が 27/29 に達することを示している．

また，Chrome の仮想認証器は（秘密鍵を含む）資格情報を平文で保存し，エクスポートが容易であるため，仮想認証器が登録される運用では，マルウェア等により資格情報が取得され，盗んだ資格情報で仮想認証器をクローンしてアカウント侵害に至り得る，というリスクを指摘している．

一方で，同研究は主にグローバルサービスを対象としており，日本の主要 Web サービスにおける導入実態や RP 側設定の傾向は十分に明らかでない．

---

[*] WebAuthn における登録要求を生成するブラウザ API．本研究における観測方法と各パラメータの定義は第 3 章で詳述する．

[†] 認証器の真正性（どの認証器モデルか等）を示す情報であり，RP が検証することで特定の認証器のみ許可するといった運用が可能になる．

[‡] WebAuthn 実装のテスト用に提供されるソフトウェア認証器．ブラウザ内部に資格情報を保持し，GUI 操作でエクスポート／インポートできる（研究では登録可否評価に利用される）．

[§] クライアント（ブラウザ）と認証器がやり取りするための通信方式．CTAP2 は FIDO2 で用いられる方式，U2F はそれ以前の方式（互換目的で残存）．

[¶] 認証器に到達するための通信経路．internal は端末内蔵（platform authenticator）を意味し，USB/BLE/NFC は外付け認証器（セキュリティキー等）を想定する．

2.3　本研究の位置づけ

本研究は，日本の主要 Web サービスを対象に，FIDO2/パスキー導入実態と RP 側 WebAuthn 設定を調査し，先行研究と同様の観測軸に基づく比較可能なデータを提示する．また，ローカル環境で PoC を構築し，仮想認証器に保存された資格情報の複製可能性を示す．

第 3 章

国内主要 Web サービスにおける FIDO2/パスキー導入実態と WebAuthn 設定調査

3.1　調査対象と手順

Tranco[2] により提供される 2025 年のサイトランキングから，ドメイン末尾が.jp であり，ブラウザから正常にアクセス可能な上位 100 サイトを抽出する．各サイトについて，トップページ，およびヘルプ・セキュリティ設定ページ等を手動で確認し，ユーザ登録，ログイン，二要素認証設定画面が存在するかを調査した．

次に，FIDO2/パスキーに対応しているかを，以下の観点から判定した．

- ログイン・セキュリティ設定画面において，「パスキー」，「FIDO2」，「セキュリティキー」，「生体認証でログイン」等の記述があるか．
- 開発者ツールの Sources タブで，`navigator.credentials.create()` および `navigator.credentials.get()` が呼び出されているか．

3.2　導入状況

表 3.1 に評価対象となる RP を特定するための測定方法の各段階における統計を示す．ブラウザで読み込み可能な上位 100 サイト中，FIDO2/パスキーに対応しているサイトは 20 件で，ユニークな RP は 17 件であった．macOS の Chrome ブラウザ環境で実際に評価用アカウントを作成し，RP 側のセキュリティ設定について調査可能なサイトは 9 件であった．

表 3.1　測定方法の各段階における FIDO2 導入件数

測定段階	数
Tranco におけるブラウザで読み込み可能なサイト	100
アカウントログイン/サインアップページのあるサイト	84
FIDO2 WebAuthn をサポートするサイト	20
FIDO2 導入サイトにおける異なる RP	17
評価可能な RP	9

3.3　RP の WebAuthn セキュリティ設定の分析

3.3.1　navigator.credentials.create() のパラメータ解析

評価可能な 9 件の各 RP について，Chrome DevTools の Sources タブにおいてパスキー登録時に `navigator.credentials.create()` に渡される `publicKey` オブジェクトの内容を調査した．以下の 3 つのパラメータを調査対象とした．

- authenticatorSelection.authenticatorAttachment：登録を許可する認証器のタイプを制限するためのパラメータ
  - “platform”（端末内蔵認証器のみ許可），“cross-platform”（セキュリティキー等の外付け認証器も許可）
- attestation：認証器の真正性を証明するための情報を要求するかどうかを決めるパラメータ
  - “none”（attestation 不要），“indirect”，“direct”
- authenticatorSelection.userVerification：パスキー登録/認証時に認証器によるローカルユーザー検証 (PIN や生体認証などによる本人確認) を要求するかどうかを決めるパラメータ
  - “required”，“preferred”，“discouraged”

3.3.2　仮想認証器の登録可否実験

さらに，Chrome DevTools の WebAuthn パネルで提供される「仮想認証器」を用いて，各 RP がテスト用認証器の登録を許容するかを評価した．仮想認証器は，本来 WebAuthn 実装の動作確認用に用意されたソフトウェア認証器であり，資格情報（秘密鍵を含む）をソフトウェアとしてブラウザ内部に保持し，GUI 操作でエクスポート/インポートできるものである．実サービスにおいてこのような認証器を登録可能とすると，マルウェアや悪意ある拡張機能によって秘密鍵が抽出され，なりすましによる認証が可能になる可能性が生じる．

本実験では，仮想認証器のプロトコル（CTAP2/U2F）およびトランスポート（internal/USB/NFC/BLE）の組み合わせを変えながら，各 RP に対して登録を試みた．

プロトコルは，クライアント（ブラウザ）と認証器がやり取りするための通信方式を表す．CTAP2 は FIDO2 で用いられる方式であり，U2F はそれ以前の方式（互換目的で残存）である．トランスポートは，認証器に到達するための通信経路を表し，WebAuthn 仕様では usb，nfc，ble，internal などが定義されている．internal は端末内蔵（platform authenticator）を意味し，USB/NFC/BLE は外付け認証器（セキュリティキー等）を想定

する．Chrome DevTools の WebAuthn パネルでは，仮想認証器作成時に protocol（CTAP2/U2F）と transport（internal/USB/NFC/BLE）を選択できるため，本研究ではこれらの組合せを変更し，RP がどの構成を受け入れるかを評価した．

3.3.3 分析結果の概要

日本の主要 Web サービスにおける RP の WebAuthn 設定と仮想認証器受け入れ状況を表 3.2 に示す．なお，表 3.2 では，RP 名を RP1 RP9 として匿名化して示す．

本研究では，attestation が none または未指定であること，および CTAP2 の仮想認証器登録が可能であることを，テスト用仮想認証器を運用上排除しづらい状態として着目し，表 3.2 では該当箇所に下線を付した．（ここで「未指定」とは，`navigator.credentials.create()` の引数で当該パラメータが明示されていないことを指す．）

表 3.2 より，RP の WebAuthn 設定には次のような傾向が見られる．まず，authenticatorSelection.authenticatorAttachment を platform に設定して端末内蔵認証器のみに限定している RP が 5 件，パラメータを指定せず認証器の種類を制限していない RP（表 3.2 の RP2, RP7, RP8, RP9）が 4 件であった．cross-platform を明示する例はなく，日本の主要サービスでは，外付けセキュリティキーよりも OS やブラウザが提供するプラットフォーム認証器（パスキー）を主に想定していると考えられる．attestation については direct による完全な attestation を要求している RP が 3 件にとどまり，4 件は none，残り 2 件はパラメータを指定していなかった．すなわち，本調査対象の 6/9 の RP では，attestation 情報に基づく認証器の真正性検証をほとんど行っていないと考えられる．一方，authenticatorSelection.userVerification に関しては required が 5 件，preferred が 2 件であり，多くの RP がローカルな生体認証や PIN によるユーザー検証を前提とした比較的強い設定を選択していた．

仮想認証器の登録可否については，調査対象とした 9 件すべての RP が，少なくとも 1 つ以上の構成（例：CTAP2+internal，あるいは CTAP2+USB など）で Chrome の仮想認証器の登録を許容していた．内訳を見ると，5 件の RP では internal+CTAP2 構成でのみ登録が成功し，残り 4 件の RP では USB/BLE/NFC/Internal の全トランスポートに対して CTAP2 仮想認証器の登録が成功した．いずれの RP も U2F プロトコルに設定した仮想認証器の登録は受け付けておらず，国内サービスでは CTAP2 ベースのパスキー実装に移行している一方で，テスト用途の仮想認証器を実運用環境で拒否していないことが分かる．これらの傾向は，多数のグローバルサービスにおいて仮想認証器の登録が許容されていると報告した Kuchhal[1] らの結果と整合している．

表 3.2　日本の主要 Web サービスにおける RP の WebAuthn 設定と仮想認証器受け入れ状況

RP ID	RP Authenticator Preferences			RP Acceptance of Virtual Authenticator Configurations			
	Authenticator Attachment	Attestation	User Verification	USB	BLE	NFC	Internal
RP1	platform	none	required	✗	✗	✗	✓ CTAP2
RP2	—	direct	preferred	✓ CTAP2	✓ CTAP2	✓ CTAP2	✓ CTAP2
RP3	platform	direct	preferred	✗	✗	✗	✓ CTAP2
RP4	platform	none	required	✗	✗	✗	✓ CTAP2
RP5	platform	—	required	✗	✗	✗	✓ CTAP2
RP6	platform	none	—	✗	✗	✗	✓ CTAP2
RP7	—	—	—	✓ CTAP2	✓ CTAP2	✓ CTAP2	✓ CTAP2
RP8	—	direct	required	✓ CTAP2	✓ CTAP2	✓ CTAP2	✓ CTAP2
RP9	—	none	required	✓ CTAP2	✓ CTAP2	✓ CTAP2	✓ CTAP2

注. —は，RP が `navigator.credentials.create()` の引数（`publicKey` オブジェクト）で当該パラメータを指定していないことを表す.
✓ は該当構成で仮想認証器の登録に成功したことを，✗ は登録に失敗したことを表す.
下線は，本研究で「テスト用仮想認証器を運用上排除しづらい状態」として着目した箇所（例：attestation が `none` または未指定であり，かつ CTAP2 仮想認証器の登録が成功）を示す.

第 4 章

仮想認証器に対する秘密鍵クローン攻撃 PoC

4.1　目的と前提

3 節より，多くの RP が仮想認証器を登録可能であることが分かった．そこで，本節ではローカルな実験環境において，仮想認証器に保存された秘密鍵を抽出して別コンテキストに複製し，同一の FIDO2 資格情報として認証に成功する PoC を構築した．実験対象は著者自身が構築したローカルのテスト用サイトに限定し，実在サービスに対して攻撃を行わないよう十分配慮した．

4.2　パスキー認証デモサイト

PoC 構築のため，著者のローカル環境（macOS，Google Chrome）上で Node.js[3] + Express[4] を用いたパスキー認証デモサイトを実装した．Node.js は JavaScript をサーバ側で実行する実行環境であり，Express は Node.js 上で Web サーバを簡単に構築するためのフレームワークである．デモサイトでは，ブラウザが提供する WebAuthn API（navigator.credentials.create() / navigator.credentials.get()）を用いて，ユーザがパスキー登録・認証を行う．サーバ側では，WebAuthn の登録・認証はチャレンジ（使い捨ての乱数）の生成と，クライアントから返ってきた署名の検証が中心となる．これらの処理は仕様が複雑であるため，本研究では SimpleWebAuthn が提供するサーバ側ライブラリ @simplewebauthn/server を用いて実装した [5]．これは WebAuthn の検証処理を提供するライブラリであり，登録時の attestation 検証や認証時の署名検証などをまとめて行う．ユーザ・ブラウザ・RP（Web ページ／サーバ）・認証器・データベース から成る全体構成を図 4.1 に示す．また，

4.3　clone_poc.js による秘密鍵クローン

clone_poc.js は，Chrome の仮想認証器が保持しているユーザの資格情報（秘密鍵等を含む）を取得し，別の仮想認証器へ再注入することで，同一ユーザのパスキーとして認証が成立することを確認する PoC 用スクリプトである．本節では便宜上，正規ユーザを User A，攻撃者を Attacker A’ と表す．図 4.2 に PoC の概要を示す．

なお，本 PoC は攻撃者が被害者端末上でブラウザを操作できる（例：マルウェアや悪意ある拡張機能が動

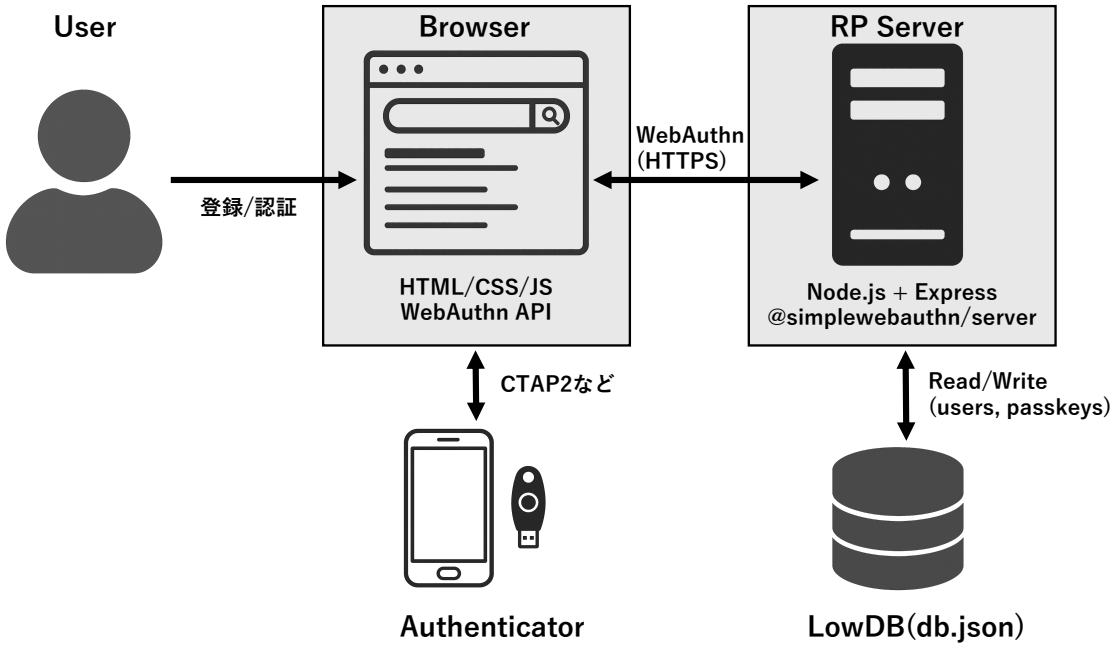

図 4.1　デモサイトのシステム構成図

作する）というローカル侵害を前提とする．

ここで用いる主要なツールと役割は次の通りである．

- Chrome DevTools（開発者ツール）
  Web ページのデバッグに使う GUI ツールであり，その中に WebAuthn パネルがある．このパネルでは，本来テスト用途の仮想認証器を作成できる．仮想認証器はハードウェアではなくソフトウェア実装であり，資格情報（秘密鍵等）をブラウザ内部に保存する．
- Chrome DevTools Protocol（CDP）
  DevTools の GUI 操作を裏側で支えている Chrome を外部プログラムから操作するためのプロトコル（API）であり，通常は人間が DevTools 画面でクリックして行う操作を，プログラムから自動で実行できる [6]．CDP には WebAuthn 関連の API 群（WebAuthn ドメイン）があり，その中の `WebAuthn.getCredentials`（仮想認証器が保持する資格情報一覧を取得）と `WebAuthn.addCredential`（取得した資格情報を別の仮想認証器へ追加）を用いることで，資格情報の抽出と再注入が可能になる [7]．
- Puppeteer[8]
  Node.js から Chrome を自動操作するためのライブラリである．ページ遷移やボタン操作の自動化に加え，CDP へ接続して API を呼び出すこともできる．本研究では Puppeteer を用いて Chrome を起動し，User A 側と Attacker A’ 側の 2 つの分離された実行環境（ブラウザの別プロファイル／別コンテキスト）を用意し，それぞれに仮想認証器を作成・操作した．

手順は下記の通りである．

(1) User A が仮想認証器 VA_A を用いて，デモサイト（RP）にパスキーを登録する．このとき VA_A 内に，

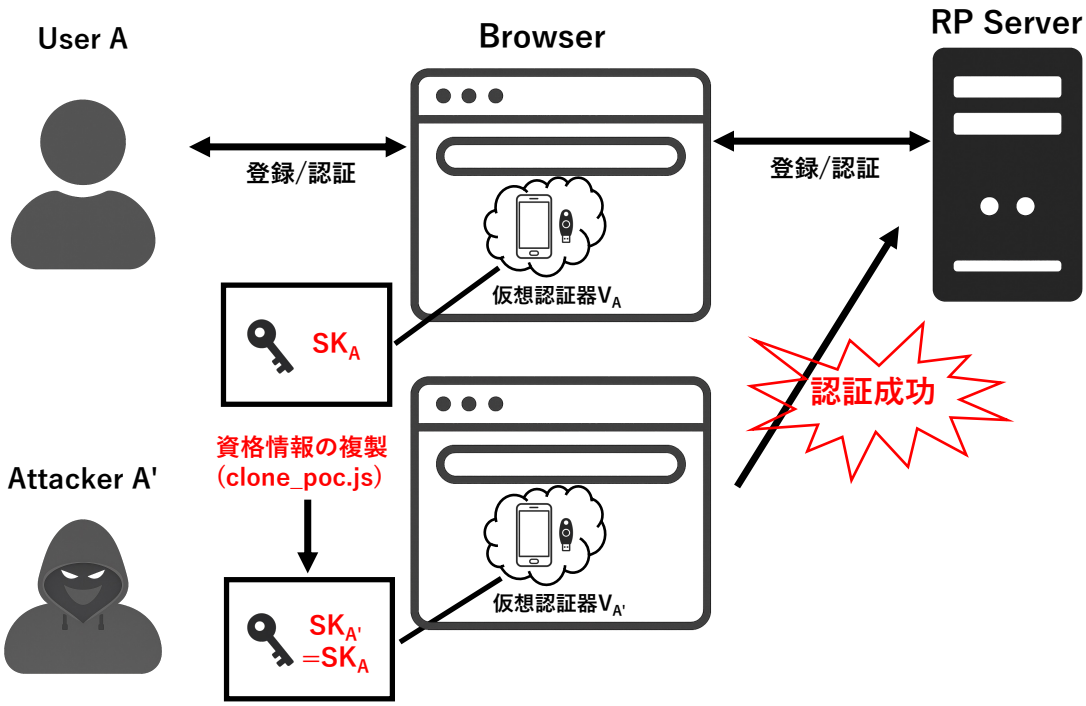

図 4.2　仮想認証器クローン PoC の概要（User A と Attacker A’）

資格情報（秘密鍵 SK_A 等を含む）が生成・保存される．

(2) 攻撃者（clone_poc.js）は CDP の `WebAuthn.getCredentials` を用いて，VA_A に保存された User A の資格情報（SK_A を含む）を取得する．

(3) 攻撃者は CDP の `WebAuthn.addCredential` を用いて，(2) で取得した資格情報を攻撃者側の仮想認証器 $VA_{A'}$ に追加し，User A の資格情報を複製する（$SK_{A'} = SK_A$）．

(4) Attacker A’ が $VA_{A'}$ を用いて User A と同一アカウントでログインを試行し，RP の検証が通過して認証が成立することを確認する．

以上により，仮想認証器に保存された資格情報が複製可能であること（＝同一のパスキーが複数環境で利用可能になること）を示した．

第 5 章

倫理的配慮と研究の制約

5.1　倫理的配慮

本研究では，Chrome 開発者ツールによる観察を中心とし，実サービスに過度な負荷を与えたり，他のユーザーに影響を及ぼしたりしないよう配慮した．調査には著者自身が作成した評価用アカウントのみを使用した．また，調査結果の公表にあたっては RP 名を匿名化して記述した．

5.2　研究の制約と限界

本研究の調査対象は，Tranco の特定時点における.jp ドメイン上位 100 サイトのうち，ブラウザからアクセス可能で，かつ評価用アカウントを作成して設定確認が可能であった RP に限定される．このため，対象集合や時期，OS・ブラウザ環境が異なれば導入状況や挙動が変化する可能性がある．また，本研究は `navigator.credentials.create()` の引数観測と仮想認証器の登録可否実験に基づくものであり，RP サーバ側の attestation 検証の実装有無を外部から完全に断定するものではない．さらに，倫理的配慮の観点から，実在サービスに対する秘密鍵抽出や不正ログイン等の攻撃は実施しておらず，PoC はローカル環境に限定した．

第 6 章

まとめ

本研究では，日本の主要 Web サイトにおける FIDO2/パスキー導入状況と RP の WebAuthn セキュリティ設定を調査した．分析の結果，多くの RP がプラットフォーム認証器（パスキー）の利用を想定していることが分かった．一方で attestation を none または未指定とする例が多数であり，認証器の真正性検証が十分に活用されていないことが分かった．userVerification については required/preferred が多く，ローカル生体認証を前提とした比較的強い設定を採用する傾向があることが確認された．

パスキー認証デモサイトと clone_poc.js を用いた PoC 実験により，仮想認証器に保存された秘密鍵がクローン可能であり，マルウェアや悪意ある拡張機能によるなりすましの認証のリスクがあることが分かった．

今後の課題としては，(i) 調査対象をスマートフォンアプリやより広範な国内サービスへ拡張すること (ii) 仮想認証器を含むソフトウェア認証器を適切に拒否・低信頼として扱うための実装例の提示などが挙げられる．

謝辞

本研究を行うにあたり，多くの方々よりご指導いただきました．特に明治大学総合数理学部先端メディアサイエンス学科，菊池浩明教授に深く感謝申し上げます．また，研究室の皆様に深く感謝の意を表するとともに，謝辞とさせていただきます．

参考文献

[1] Dhruv Kuchhal, Muhammad Saad, Adam Oest, and Frank Li. “Evaluating the Security Posture of Real-World FIDO2 Deployments”, In ACM SIGSAC Conference on Computer and Communications Security (CCS ’23). 2381–2395, 2023

[2] Tranco, (https://tranco-list.eu/，2025 年 8 月参照).

[3] Node.js, (https://nodejs.org/ja).

[4] Express, (https://expressjs.com/).

[5] SimpleWebAuthn, (https://simplewebauthn.dev/).

[6] Chrome DevTools Protocol, (https://chromedevtools.github.io/devtools-protocol/).

[7] Chrome DevTools Protocol: WebAuthn Domain, (https://chromedevtools.github.io/devtools-protocol/tot/WebAuthn/).

[8] Puppeteer, (https://pptr.dev/).

[9] えーじ，倉林雅，小岩井航介: パスキーのすべて —導入・UX 設計・実装，技術 評論社，2025.

付録 A

ウェブサイトからマウス履歴を取得するセッションリプレイサービスの検出ツールの開発

A.1　はじめに

セッションリプレイサービスは，利用者がウェブページ上で行うマウス操作やクリック，スクロールといった行動を記録・解析し，サイト改善やマーケティングに役立てる技術である．しかし，その一方で，ユーザ側からはこうしたサービスの存在を直接確認することは難しく，プライバシー面での不安が懸念されている．梶間らの研究 [1] によると，多くのユーザは自分のウェブ上での行動が外部の追跡サービスによって記録・利用されていることを十分に認識していないと報告されている．

そこで，本研究では，ウェブサイトに導入されているセッションリプレイサービスを自動的に検出するツール (以後検出ツールと呼ぶ) を開発し，その精度を評価した．まず自分で用意したウェブサイトにセッションリプレイサービスを仕込み，その挙動を確認した上で，検出手法を提案・実装した．さらに，実際の主要なウェブサービスを対象に検出精度を評価し，提案手法の有効性を検証した．

表 A.1　トップウェブサイトにおけるセッションリプレイサービスの導入状況（全サイト数を母数）

	世界 (70 サイト)		日本 (70 サイト)	
サービス名	数	%	数	%
Microsoft Clarity	12	17.1	12	17.1
Hotjar	2	2.9	6	8.6
Mouseflow	0	0.0	0	0.0
Yandex Metrica	2	2.9	0	0.0
Contentsquare	0	0.0	0	0.0
Crazyegg	1	1.4	1	1.4
Dynatrace	0	0.0	1	1.4
FullStory	1	1.4	3	4.3
LogRocket	0	0.0	0	0.0
Lucky Orange	0	0.0	0	0.0
合計	18	25.7	23	32.9

A.2　セッションリプレイサービスの導入状況調査

A.2.1　調査方法

ウェブサイト上のトラッキングスクリプトを検出し，サイトが使用しているトラッキング技術やツールを排除するブラウザ拡張である，Ghostery[3] を用いて，ウェブサイトに導入されているセッションリプレイサービスを手動で調査した．

A.2.2　調査対象

トップウェブサイトランキングを提供している Tranco[4] の 2024 年のデータセットの内，事前にページが閲覧可能な世界の上位 70 サイト，および，日本の上位 70 サイトのリストを調査対象とした．検出対象のセッションリプレイサービスは，Microsoft Clarity[5]，Hotjar[6]，Mouseflow[7]，Yandex Metrica[8]，contentsquare[9]，Crazyegg[10]，Dytatrace[11]，fullstory[12]，luckyorange[13]，logrocket[14] の 10 である．

A.2.3　調査結果

表 A.1 にセッションリプレイサービスの導入率を示す．日本のサイトの方が導入が進んでいる．世界の上位 70 サイトに載っているサイトの多くが日本の上位 70 サイトにも載っていることを考慮すると，世界の上位を占めるトップウェブサイトにおいて，それほどセッションリプレイサービスの導入が進んでいないことが推察される．

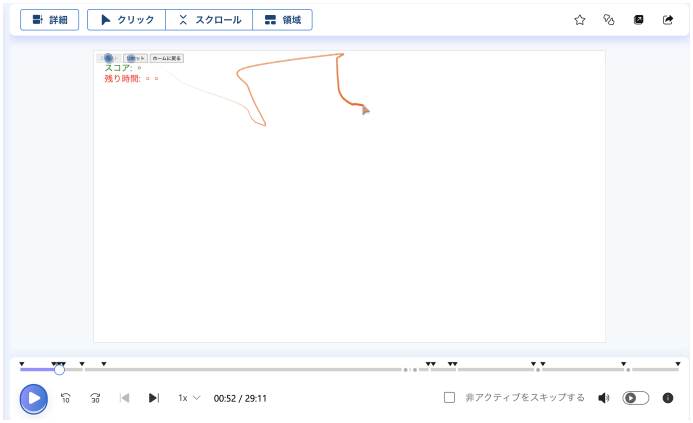

図 A.1　デモ画面例

A.3　セッションリプレイサービス検出ツールの開発

A.3.1　概要

本研究では，Web ページ上に埋め込まれているセッションリプレイサービス特有のコードパターンを自動的に抽出し，当該サイトがいずれかのセッションリプレイツールを導入しているかを判別するシステムを開発した．本システムでは，以下の原理と手順に基づいて検出する．

まず, ウェブサイトにセッションリプレイサービスを仕込み，その挙動を確認した. 図 A.1 に，取得したマウス履歴のデモ画面を示す.

セッションリプレイサービスはサービス提供者ごとに異なる JavaScript コードや特定のスクリプト URL を用いている．本システムはこれらサービス固有のコードパターン（例えば，`clarity.ms/tag/` など）を定義しておく．対象サイトのソースコード中にこれらのパターンが出現すれば，対応するセッションリプレイサービスが導入されていると判定可能する．

処理の流れは以下の通りである．

(1) 事前に用意した CSV ファイルから解析対象サイトの URL リストを読み込む.
(2) Selenium WebDriver と ChromeDriver を用いて自動的に各サイトへアクセスし，取得した HTML から `<script>` タグを抽出する.
(3) 本システムは，各 `<script>` 要素の `src` 属性や内部コードを正規表現マッチングにより解析し，定義済みのパターンが検出された場合は対応するサービス名を出力する.

本システムは Python を用いて実装している．システム構成図を図 A.2 に示す．また，本研究では Ghostery による検出結果を参考に正解データを定め，本システムの同一サイト群に対する検出結果の精度を評価した．指標は Precision（適合率），Recall（再現率），F1 値を評価する．

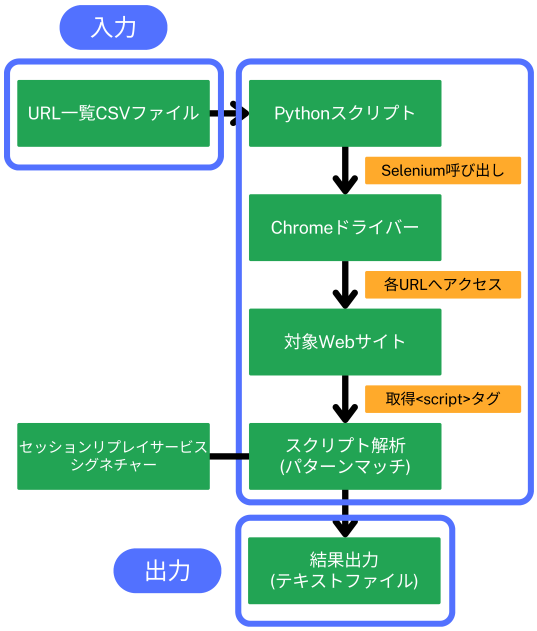

図 A.2　システム構成図

表 A.2　検出ツールの精度指標

	Precision	Recall	F1
世界	0.769	0.556	0.645
日本	0.778	0.304	0.438
平均	0.774	0.43	0.542

A.3.2　結果

検出ツールの精度を表 A.2 に示す．実験の結果，同じサイトに複数のセッションリプレイサービスが導入されているケースにおいて，サービスごとに区別すると FN については世界で 8 件，日本で 16 件であった．また，FP については世界で 3 件，日本で 2 件であった．

A.3.3　誤検出，未検出の例と原因の考察

FN となったサイトの例としては，https://www.bing.com/ や https://unity.com/ja などが挙げられる．このようなサイトに対して手動でトップページのソースコードを調べたところ，定義済みのパターンと一致する文字列は見受けられなかった．原因としては，トップページ以外のページにセッションリプレイサービスが導入されている場合や，本実験で使用したパターンと異なる JavaScript コードや特定のスクリプト URL を用いている場合などが考えられる．

FP となったサイトの例として，https://wordpress.com/ja/を挙げる．このサイトに対して手動でトップページのソースコードを調べたところ，図 A.3 のように定義済みのパターンと一致する文字列 (clarity.ms/tag/) が見つかった．よって検出ツールでは Microsoft Clarity が導入されていると判定したが，Ghostery ではいずれ

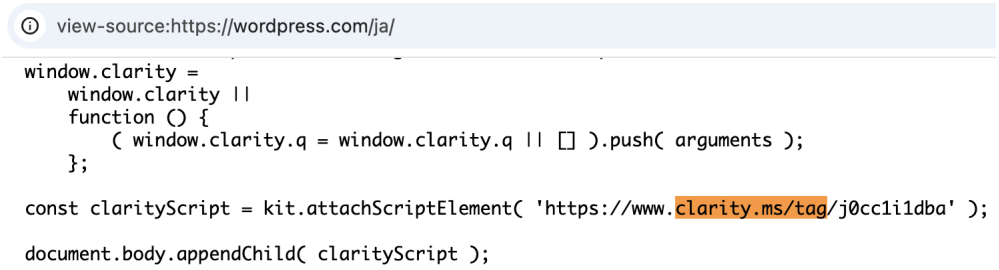

図 A.3　wordpress.com のソースコードの一部

のセッションリプレイサービスも検出されなかった．原因として，この Clarity は昔は使われていたが，現在は動作していない可能性が考えられる．

A.4　おわりに

本研究では，国内外の主要なウェブサイトにおけるセッションリプレイサービスの導入状況を調査し，セッションリプレイサービス検出ツールを開発した．平均 F1 で 0.542 の精度を評価した．今後の課題として，まず，実験で使用した URL リストを対象に，各ウェブサイトで実際にセッションリプレイサービスが利用されているかを手動で確認する作業を挙げる．そして得られた結果を基準データとし，これをもとに，Ghostery と本研究で開発した検出ツールの検出結果を比較し，それぞれの精度を評価する．さらに，本ツールの精度が低い原因を詳しく調査し，改善点を特定する．

参考文献

[1] 梶間大地，菊池浩明，“セッションリプレイサービスからの個人識別性と国内外サイトにおけるプライバシーポリシーでの公表状況”，第 104 回 CSEC 研究発表会，2024-CSEC-104，No.11，pp1-8，IPSJ，2024.

[2] Selenium，(https://www.selenium.dev/ja/ documentation/，2024 年 8 月参照).

[3] Ghostery，(https://www.ghostery.com/，2024 年 8 月参照).

[4] Tranco，(https://tranco-list.eu/，2024 年 8 月参照).

[5] Microsoft Clarity，(https://clarity.microsoft. com/，2024 年 8 月参照).

[6] Hotjar，(https://www.hotjar.com/，2024 年 8 月参照).

[7] Mouseflow，(https://mouseflow- jp.com/，2024 年 8 月参照).

[8] Yandex，(https://metrica.yandex.com/about?，2024 年 8 月参照).

[9] Contentsquare，(https://contentsquare.com/ jp-jp/，2024 年 8 月参照).

[10] Crazyegg，(https://www.crazyegg.com/，2024 年 8 月参照).

[11] Dynatrace，(https://www.dynatrace.com/ja/，2024 年 8 月参照).

[12] fullstory，(https://www.fullstory.com/，2024 年 8 月参照).

[13] lucky orange，(https://www.luckyorange.com/，2024 年 8 月参照).

[14] LogRocket，(https://logrocket.com/，2024 年 8 月参照).