

Tracking for Good: Finding Behavioral Biometrics on the Web using Static Taint Analysis

by

Aswad Tariq

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2025

© Aswad Tariq 2025

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Behavioral biometric technologies have emerged as powerful tools for enhancing digital security by analyzing unique user interactions such as keystrokes, mouse movements, and touch gestures. This thesis provides a systematic exploration and empirical measurement of behavioral biometric scripts deployed across the web, particularly focusing on their prevalence and implications in critical interactions such as user authentication and fraud prevention.

Our comprehensive approach begins with manual and automated identification and characterization of scripts from major behavioral biometric providers including BioCatch, BehavioSec, TransUnion’s Iovation, and Mastercard’s NuData, among others. Leveraging an advanced static taint analysis framework utilizing Visible V8, we effectively trace behavioral biometric data flows within JavaScript, accurately identifying sensitive data collection and transmission points. To reliably detect login webpages containing behavioral biometric scripts, we developed LoginGPT, a state-of-the-art web crawler enhanced by Large Language Models (LLMs), significantly outperforming existing heuristic-based solutions in identifying login pages.

Furthermore, we develop a supervised machine learning approach using Random Forest classifiers trained on vendor-agnostic static analysis features, achieving robust accuracy and strong generalization to previously unseen vendors. Our comprehensive empirical evaluation spans 9,502 U.S. banking websites and the Chrome User Experience (CrUX) top 100,000 domains, revealing that behavioral biometric scripts are deployed on **15.8%** of banking domains with discoverable login pages and **1.79%** of general web domains with discoverable login pages. Our findings demonstrate the strategic deployment of these technologies on high-risk interfaces such as authentication pages, uncover distinct vendor deployment patterns across industries, and highlight significant privacy concerns stemming from extensive behavioral data collection practices.

This thesis contributes a robust framework and critical insights for detecting, characterizing, and understanding behavioral biometric technologies on the web, offering valuable perspectives for researchers, industry professionals, and policymakers engaged in digital security and privacy protection.

Acknowledgements

I would like to thank everyone who made this thesis possible.

First and foremost, I am deeply grateful to my advisor Dr. Urs Hengartner for his unwavering guidance and support throughout this journey. I also extend my sincere appreciation to Alexandru Bara for his constant encouragement and invaluable critical feedback that helped shape this work.

I am equally thankful to the friends who became family during this process: Haseeb-ur-Rehman Faheem, Mehdi Aghakishiyev, Abdelkarim Kati, Haseeb Ahmed, Lucas Fenaux, and many others who offered encouragement and motivation when I needed it most. Their support and belief in my work made all the difference.

Finally, I acknowledge everyone else who contributed to this thesis in ways both big and small, your impact has not gone unnoticed.

Dedication

This is dedicated to my parents, my family and the one I love.

Table of Contents

Author's Declaration	ii
Abstract	iii
Acknowledgements	iv
Dedication	v
List of Figures	xi
List of Tables	xii
1 Introduction	1
1.1 What Are Behavioral Biometrics	1
1.2 Applications of Behavioral Biometrics	1
1.3 Goals Of This Study	2
1.4 Methodology	3
1.4.1 Initial Behavioral Biometric Script Collection	3
1.4.2 Static Taint Analysis	3
1.4.3 Login Crawler	3
1.4.4 Machine Learning Model	3
1.5 Contributions & Implications	4
1.6 Thesis Structure	5

2	Behavioral Biometric Script Collection	6
2.1	Background	6
2.2	Related Work	8
2.3	Script Identification and Categorization Methodology	8
2.3.1	Company Identification	9
2.3.2	Client Finding	10
2.3.3	Script Identification and Classification	10
2.4	Behavioral Biometric Scripts	11
2.4.1	Mastercard NuData NuDetect	11
2.4.2	TransUnion Iovation	12
2.4.3	BioCatch	12
2.4.4	LexisNexis BehavioSec	14
2.4.5	Ping Identity PingOne	14
2.4.6	D4T4 Celebrus	16
2.4.7	Forter	16
2.4.8	Bot Detectors	17
2.4.9	Other Companies	18
2.5	Conclusion	20
3	Static Analysis in JavaScript	24
3.1	Introduction	24
3.2	Related Work	25
3.2.1	Static Analysis for Browser Fingerprinting Detection	25
3.2.2	Dynamic Analysis for Browser Fingerprinting Detection	26
3.2.3	Our Hybrid Approach	27
3.3	Design	27
3.3.1	VisibleV8	27
3.3.2	PDG & JStep	29

3.3.3	Taint Analysis	32
3.3.4	System Design and Scalability	35
3.4	Limitations	36
3.5	Conclusion	39
4	Login Page Crawler	40
4.1	Introduction	40
4.2	Related Work	41
4.2.1	Heuristics-Based Login Crawlers	41
4.2.2	Vision Language Models for Web	42
4.3	LoginGPT	43
4.3.1	Login Page Detection Strategies	44
4.3.2	LLM Crawler	44
4.3.3	Classification Model	48
4.4	Evaluation	48
4.4.1	System Configuration	48
4.4.2	Bank Login Pages	49
4.4.3	CrUX Top 1000	50
4.5	Conclusion	52
5	Behavioral Biometric Script Classification	54
5.1	Introduction	54
5.2	Dataset and Labeling	55
5.3	Classification Methodology	56
5.3.1	Heuristics Based Classification	57
5.3.2	Feature Engineering and Selection Methodology	58
5.3.3	Supervised Training Methodology	64
5.3.4	Results	71

5.4	Final Model	74
5.5	Discussion	75
5.6	Conclusion	77
6	Evaluation	78
6.1	Introduction	78
6.2	Methodology	79
6.2.1	Dataset Construction & Justification	79
6.2.2	Crawling & Classification Pipeline	80
6.3	Results	82
6.3.1	System Configuration	82
6.3.2	U.S. Banking Sector	83
6.3.3	CrUX Top 100K Domains	86
6.3.4	Landing Versus Login Study Comparison	90
6.3.5	Comparison with Dynamic Taint Analysis Study	91
6.4	Discussion	94
6.5	Conclusion	97
7	Conclusion	99
7.1	Key Contributions	99
7.2	Important Findings	100
7.3	Future Work	101
	References	102
	APPENDICES	118

A	Login Crawler Prompts	119
A.1	Model Prompts	119
A.1.1	Cookie Banner Detection Prompt	120
A.1.2	Login Element Detection Prompt	120
A.1.3	Classification of Login Page Prompt	121
A.2	CrUX Top 1000 Domains Crawl Success	121
B	Classifying Behavioral Biometric Scripts	123
B.1	Keywords For Known Companies	123
B.2	Model Hyperparameters	124
B.2.1	Optimal Hyperparameters	124
C	Taint Analysis Sources and Sinks	129
C.1	Source APIs	129
C.1.1	Behavioral Source APIs	129
C.1.2	Fingerprinting Source APIs	129
C.1.3	Sink APIs	129

List of Figures

2.1	NuDetect Collection Flow	11
2.2	BioCatch Collection Flow	13
2.3	This image published by Experian [48], shows the landscape of fraud prevention vendors and their capabilities [47]. Note that many, but not all, of these companies provide behavioral biometric services for fraud prevention.	21
3.1	Simple PDG for the program in Listing 3.4	31
3.2	Taint Analysis System Diagram	35
3.3	Obfuscated AST vs Unobfuscated AST	38
4.1	LoginGPT system diagram	43
4.2	Flowchart showing crawler’s internal logic	45
4.3	Domain Coverage Overlap Between Login Page Detection Systems	52
4.4	Distribution of LoginGPT detection strategies across analyzed domains. . .	53
6.1	Evaluation pipeline system diagram	81
6.2	NuData authentication provider user interface across different webpages . .	94

List of Tables

2.1	Behavioral Biometric and Fraud Prevention Companies by Category	22
2.2	Behavioral Biometric and Fraud Prevention Companies by Category (continued)	23
4.1	Comparison of Login Page Detection Systems on 50 random US bank websites	49
4.2	Summary of Domain Accessibility in 1000 CrUX Websites	50
4.3	Login Page Detection Performance by System	51
5.1	Script Classification Raw Features and Descriptions	55
5.2	Complete Set of Vendor-Agnostic Features for Behavioral Biometric Script Detection	58
5.3	Top Twelve Important Features for Behavioral Biometric Script Detection (Balanced Dataset see Section 5.3.3)	62
5.4	Top Twelve Features for Behavioral Biometric Script Detection (Imbalanced Dataset, see Section 5.3.3)	63
5.5	Model Evaluation Results (Imbalanced Dataset)	72
5.6	Model Evaluation Results (Balanced Dataset)	72
6.1	Website Reachability Analysis (N = 9,502 bank websites)	83
6.2	Behavioral Biometric Scripts Distribution for 9,502 Banks via Landing-Pages	84
6.3	Login Page and Behavioral Script Analysis	85
6.4	Behavioral Biometric Scripts Distribution for 9,502 Banks via Login-Pages	86
6.5	Large-Scale Website Reachability Analysis (N = 100,000 domains)	87

6.6	Behavioral Biometric Scripts Distribution for CrUX Top 100K via Landing-Pages	89
6.7	Behavioral Biometric (BB) Adoption Across Studies and Page Types	90
6.8	Behavioral Biometric Script Prevalence Comparison	91
6.9	Behavioral Biometric Vendors Identified via Login Pages (N = 53,308 domains)	98
A.1	Domain Success and Failure Analysis of 1000 Crux Websites	122
B.1	Security and Fraud Detection Companies and Associated Keywords	123
B.2	Optimal Hyperparameters (Imbalanced Dataset)	125
B.3	Optimal Hyperparameters (Balanced Dataset)	126
C.1	Known data sinks monitored for behavioral biometric data exfiltration	130

Chapter 1

Introduction

1.1 What Are Behavioral Biometrics

Behavioral biometrics are a method of user authentication and identification based on analyzing unique patterns in user interactions with digital devices. Unlike traditional biometric methods that rely on physical characteristics, such as fingerprints, iris scans, or facial recognition, behavioral biometrics leverage behavioral patterns including keystroke dynamics, mouse movements, touch gestures, and device handling patterns (e.g., accelerometer and gyroscope data). These patterns are typically subconscious, consistent, and challenging to imitate, making them effective indicators of a user's identity.

Keystroke dynamics analyze timing information associated with typing, such as dwell times (how long a key is pressed) and flight times (intervals between keystrokes). Mouse dynamics examine cursor movements, click patterns, and scrolling behavior, while touch dynamics assess the pressure, speed, and trajectories of touch interactions on touchscreen devices. These nuanced behaviors create robust user profiles that continuously authenticate users throughout a session, offering both security and convenience.

1.2 Applications of Behavioral Biometrics

Behavioral biometrics find diverse applications across several critical domains due to their continuous and unobtrusive authentication capabilities. Financial institutions widely adopt behavioral biometrics to combat account takeover attacks, credential stuffing, and fraudulent transactions [102, 158]. In this context, behavioral biometrics enhance existing security

measures by monitoring user behavior during online banking activities, flagging anomalous interactions indicative of unauthorized access attempts.

In e-commerce, these techniques help differentiate legitimate customer interactions from automated bots or malicious actors, preventing fraud while preserving user experience. Additionally, behavioral biometrics support anti-bot technologies on platforms vulnerable to automated abuse, such as ticket sales, online gaming, gambling websites, and dating platforms.

Other sectors leveraging behavioral biometrics include healthcare—protecting patient data access; enterprise—securing remote access and internal applications; and government—ensuring secure authentication for sensitive digital services. Behavioral biometrics are thus instrumental in safeguarding digital identities and enhancing security without compromising user convenience.

1.3 Goals Of This Study

The overarching goal of this study is to systematically measure the prevalence and adoption patterns of behavioral biometric services on the modern web. Specifically, we aim to:

- Quantify the presence of behavioral biometric scripts across a comprehensive dataset, focusing specifically on the financial sector and broader internet use.
- Identify which behavioral biometric vendors dominate the market and the types of websites that adopt these technologies.
- Examine the extent of deployment differences between login and landing pages, hypothesizing that critical interactions such as user authentication are primary targets for behavioral biometrics.
- Provide detailed insight into the global prevalence of behavioral biometrics using the Chrome User Experience (CrUX) top 100K domains list [\[41\]](#) and a targeted analysis of 9,502 U.S. banking websites.

1.4 Methodology

1.4.1 Initial Behavioral Biometric Script Collection

Our study begins with an extensive manual and automated effort to identify behavioral biometric scripts. Initially, we manually compile a dataset from known behavioral biometric vendors by analyzing their client websites and identifying specific scripts via distinctive patterns, behaviors, and script sources. This foundational dataset comprises 232 behavioral biometric scripts from 18 unique vendors, establishing a solid baseline for further analysis.

1.4.2 Static Taint Analysis

To systematically identify and classify scripts at scale, we employ static taint analysis using a specialized instrumentation framework called VisibleV8. This framework modifies the Chrome browser engine, allowing detailed observation and analysis of JavaScript execution. Our static analysis approach traces data flows from behavioral source APIs through intermediate processing steps to sensitive sinks, enabling precise determination of scripts collecting and aggregating behavioral biometric data. The static taint analysis is vital for automated detection and accurate characterization of script behaviors, significantly reducing reliance on manual verification.

1.4.3 Login Crawler

To locate critical pages—particularly login pages, hypothesized to contain a higher concentration of behavioral biometric scripts—we design and implement LoginGPT, a state-of-the-art web crawler enhanced by Large Language Models (LLMs). LoginGPT utilizes visual and textual page content, leveraging vision-language models to precisely detect login functionalities. This advanced crawler demonstrates significantly superior precision compared to previous login page detection techniques, reliably identifying login workflows at large scale across diverse website designs and structures.

1.4.4 Machine Learning Model

We employ supervised machine learning to classify behavioral biometric scripts accurately. Utilizing vendor-agnostic features identified via static taint analysis—including API usage

diversity, data aggregation behaviors, and interaction complexity—we train a Random Forest classifier and other supervised models. Through rigorous nested cross-validation and careful handling of vendor-aware data splits, our model achieves high accuracy, generalizes effectively to unseen vendors, and successfully distinguishes behavioral biometric scripts from similar scripts used for analytics or user experience enhancements.

1.5 Contributions & Implications

Our study provides several notable contributions to the fields of web security and privacy measurement:

- We developed and evaluated LoginGPT, an innovative LLM-driven crawler demonstrating significantly improved login page detection capabilities compared to existing crawlers like SSO-Monitor [83] and Double-Edged Sword [144], enabling precise large-scale measurements.
- We present the first comprehensive large-scale empirical measurement of behavioral biometric prevalence on the web, analyzing 9,502 U.S. bank websites and the top 100,000 websites from the CrUX dataset.
- We introduce a robust methodology combining static taint analysis and machine learning to accurately detect and classify behavioral biometric scripts, significantly reducing manual analysis overhead.
- We identify and document five, previously unknown to us, behavioral biometric providers through our systematic analysis, further expanding the understanding of the behavioral biometric vendor landscape.
- Our findings highlight significant privacy and security implications, emphasizing the prevalence of behavioral biometrics in critical user interactions and raising awareness about potential privacy concerns related to continuous behavioral data collection.

These contributions provide actionable insights for web security practitioners, privacy advocates, and policymakers, guiding informed decisions about behavioral biometric deployments and regulatory considerations.

1.6 Thesis Structure

This thesis is structured to clearly articulate each stage of our research:

- Chapter 2 provides foundational background on behavioral biometrics, detailing the technologies, use cases, and current challenges in script detection and privacy implications. It also describes our initial script collection methodology and manual labeling effort, setting the stage for automated detection.
- Chapter 3 outlines our static taint analysis framework using VisibleV8, describing the instrumentation and data-flow tracing methodologies critical to our automated detection.
- Chapter 4 presents LoginGPT, our advanced crawler design, highlighting its superior detection capabilities and precision in identifying login workflows.
- Chapter 5 discusses the supervised machine learning approach, detailing feature engineering, model selection, and performance validation to reliably classify behavioral biometric scripts.
- Chapter 6 provides comprehensive empirical results from our large-scale studies on the prevalence of behavioral biometrics across U.S. banking websites and the broader web landscape.
- Chapter 7 collects our findings, discusses their implications for web security and privacy, acknowledges limitations, and suggests avenues for future research.

This structured approach ensures clarity and coherence, progressively guiding the reader through our research, methodologies, findings, and their broader implications.

Chapter 2

Behavioral Biometric Script Collection

The following chapter will discuss how we collected and categorized scripts for companies that provide behavioral biometric services. These scripts will be used as ground truth for the remainder of this thesis.

2.1 Background

Modern web applications, at their core, are composed of Hyper Text Markup Language (HTML), Cascading Style Sheets (CSS), and JavaScript. HTML outlines the core structure of a webpage and the hierarchy of its elements (parent and child relationships between elements). CSS defines the style of these elements, what color each button should be, the positioning of an element on the page, etc. Finally, JavaScript is what makes a page *functional*. Functionality refers to the responsiveness to a page for clicks (opening up a modal when a button is pressed), submitting login forms, etc. This can also be used for higher-level tasks, such as collecting the browser fingerprint for a user. This is done by collecting certain information about the user’s browser via specific browser APIs (e.g., `Navigator.plugins`, `Navigator.userAgent`, etc.) [175, 95]. These browser fingerprints have been historically used for nefarious purposes, such as stateless tracking of users across the web [106]. Stateful tracking refers to the practice of storing a unique identifier in browser cookies—or other client-side storage—so that a tracker can recognize and follow a user’s activity across multiple visits and websites [1, 96, 58, 149]. This can then be used to track

users over time. This data can be removed by users, and recent constraints by the GDPR have forced website providers to take user consent before applying any cookies [98]. Hence, we have seen a shift to stateless tracking techniques like fingerprinting, that take information about a user’s browser and exfiltrate it from the webpage. Using this information the website provider can construct a unique identifier for each user, which can then be used to track them across the internet [46, 1]. These fingerprints may be used to show the user relevant ads or to offer them a personalized website experience [106]. Contrastingly, they can also be used to enhance the security of a webpage, such as bot detection, and fraud prevention [169, 28, 132, 59]. Organizations may also use these fingerprints to flag fraudulent IP addresses across the web. This prevents Denial Of Service (DOS) attacks, as well as finding if an account is being logged into from two different devices in two different continents at the same time [34]. Therefore, fingerprinting has numerous privacy implications. Deciding whether the positive applications (security, bot detection) outweigh the negative (tracking for advertisements) is not in the scope of this thesis.

Fingerprints can be used to find if two different devices, in different locations, are logging into the same account at similar times, but they cannot prevent a more advanced attack: where the user’s device is compromised physically or the browser fingerprint for the user is spoofed [117] or masked using browsers like Brave and Tor [98]. Given an attacker with the user’s browser fingerprint and login credentials (possibly obtained via a past data leak), they can access the user’s sensitive accounts. Additionally, if the attacker has access to credit card information, they can successfully use it to execute fraudulent payments on the user’s behalf.

Behavioral biometrics provides an effective solution to mitigate credential-based attacks by analyzing unique patterns in user interactions. Specifically, we define behavioral biometrics as the usage of keystroke, mouse, touch, accelerometer, gyroscope, and other sensory data to identify users. This also includes walking (gait) patterns (we do not consider voice as a behavioral biometric). Once gathered, this data is transmitted securely to backend servers, where sophisticated machine learning models are employed to detect anomalous activities. The models analyze real-time interaction patterns against established behavioral baselines, distinguishing legitimate user behavior from potentially fraudulent activities that indicate credential misuse or unauthorized account access attempts.

In this chapter, we systematically identify and analyze JavaScript-based behavioral biometrics scripts deployed by various vendors across the web. Our approach involves manually collecting these scripts through targeted web crawling and in-depth analysis. We utilize vendor documentation, industry whitepapers, and identifiable patterns in data collection methodologies to classify and validate these scripts accurately. By leveraging this best-effort approach, we aim to comprehensively document the prevalence, characteristics, and

techniques utilized by major behavioral biometrics providers, thus providing a foundation for understanding their behavior and data collection patterns.

2.2 Related Work

As mentioned in Section 2.1, behavioral biometrics are being used on the internet today, but to the best of our knowledge, no study has attempted to measure the prevalence of these solutions on the web. While this is true, some papers that focus on measuring browser fingerprinting have indirectly come across some scripts that are used for behavioral biometrics [144, 40, 10, 80, 151, 114, 106]. Most of these mention bot detection solutions such as Datadome [35] and Human/PerimeterX [76], as well as notable behavioral biometric providers, NuData [118] and Iovation [79]. Some of these papers do not perform a detailed analysis of the vendor associated with each script. Their collected data contains scripts for vendors that provide behavioral biometric services [144, 40, 80, 151]. These vendors include Behaviosec [101], Biocatch [15] and CallSign [21].

Additionally, there is another paper by Prognov et al. [130] that mentions existing solutions from popular fraud prevention providers such as NuData [118], SecuredTouch [141], and Zighra [176] that offer behavioral biometric services for mobile devices.

To the best of our knowledge, the first commercial usage of behavioral biometrics was by The Royal Bank of Scotland in 2016 [120]. They employed the services of Israeli startup Biocatch [15], to counter account takeover fraud [23] and credential stuffing attacks [44]. The company has since been acquired by Permira [121], but still remains one of the top fraud prevention providers to many large commercial banks. Since then, Biocatch and many other notable players have created a multibillion-dollar industry [113, 150, 70, 136]. This market is predicted to reach 10.5 billion USD by 2033 [77]

In the following section, we will discuss our approach to identifying companies that provide behavioral biometrics solutions on the web.

2.3 Script Identification and Categorization Methodology

At the beginning of our study, we learned from Mastercard about their recently acquired behavioral biometric company NuData Security [109]. We then attempted to find other

comparable products. Our approach can be considered top-down, where we first attempt to find companies that provide behavioral biometric services and then attempt to find clients for these companies. From these clients, we attempt to find scripts that are associated with the behavioral biometric companies using the methods mentioned below. This was a manual effort by two researchers over six weeks, where we analyzed and categorized scripts from more than 40 companies [11].

2.3.1 Company Identification

To identify behavioral biometric companies/vendors, we used a variety of methods. First, we identified what companies provide these services and solutions. We took NuData as a seed and looked for competitors for this company and any other companies that mentioned behavioral biometrics on their websites. Simultaneously, we looked at recent market research reports about the behavioral biometric industry [70, 77]. We also checked online articles and company whitepapers to identify behavioral biometric companies [156, 31, 17]. It is important to note that we were specifically looking for behavioral biometric providers for the web, and while we found some behavioral biometric companies that provide services for mobile devices, we do not claim to have comprehensive coverage for those providers [176, 141]. To confirm if a company was providing behavioral biometric services, we attempted to use the following three approaches in order:

1. Reaching out to representatives of companies that we suspected may have behavioral biometric products.
2. Reading company whitepapers to understand if they had any products that are similar to our definition of behavioral biometrics.
3. Reading through company documentation (if available) to understand their products and APIs.

The most challenging part was finding more companies that may provide behavioral biometric services. We primarily focused on companies that provided fraud prevention services and leveraged these keywords (e.g., fraud prevention, account takeover) to reduce our target set of companies that needed to be checked by the methods mentioned above. Using these methods, we found 39 companies shown in Table 2.1 and Table 2.2. The table lists companies for which we found scripts and those for which we did not find scripts. The “Script Active” column says we found this script actively collecting data on a website.

We used a best-effort methodology to confirm our suspicion of a company doing behavioral biometrics (as mentioned above). We do not consider any other source when classifying a script as behavioral biometric or not. We believe this is a fair measure meant to make sure the companies we analyze in our study truly do provide behavioral biometric services, and we do not focus on companies that do not make these claims themselves.

2.3.2 Client Finding

Next, we focused on finding clients for each company. We do this using two main methods. First, some companies directly mention a list of clients that use their services. Even if these client lists are not up to date, they gave us insights into which companies are willing to pay for behavioral biometric services. So often, using client lists for some companies that were not up to date, we found scripts for other companies. Second, using the website, whitepapers, and documentation for companies, we extract keywords and use these to search for scripts from the behavioral biometric company on the web. To search for these keywords on JavaScript on the web, we use NerdyData [116], a tool for large-scale web code search. Using this method, we found multiple clients for many behavioral biometric companies. We use these clients to construct the ground truth for behavioral biometric scripts for our study.

2.3.3 Script Identification and Classification

For clients of behavioral biometric companies, we are still left with the task of finding the correct script for each company on the client’s web page. We do this by searching for keywords in scripts and finding notable patterns among scripts of interest. (e.g., `wupServerURL` for Biocatch and `ndjsStaticVersion` for NuData). We look at scripts that listen to mouse, keystroke, and touch events. Behavioral biometric scripts often send periodic data to the backend, which is similar to analytics and session replay scripts. In some cases, these scripts have copyright text that directly links them to a company. Some scripts, however, are obfuscated (modified to be unreadable), meaning normal keywords will not work. For such scripts, we look at the outgoing URL for script data, such as `accdab.net` for Accertify [3].

Using an instrumented version of Chrome called VisibleV8 [86] (a browser that shows raw browser API collection information for each JavaScript file on a webpage), we check the collected APIs from each JavaScript file on client websites. We notice that behavioral

biometric scripts collect more behavioral and fingerprinting APIs than other scripts, for example, `MouseEvent.pageX`, `Navigator.plugins`, and `KeyboardEvent.keyCode`.

2.4 Behavioral Biometric Scripts

In this section, we discuss some notable behavioral biometric scripts, information about how we found these scripts, and how data flows within these scripts. We will use this as background in the following sections to justify our design decisions for the components of our study. The goal with analyzing data flow for each script was to identify unique features of these scripts against others, especially versus other scripts that collect behavioral data as well as browser fingerprints, such as Clarity [112], HotJar [74], Clicktale [30], and FullStory [60].

2.4.1 Mastercard NuData NuDetect

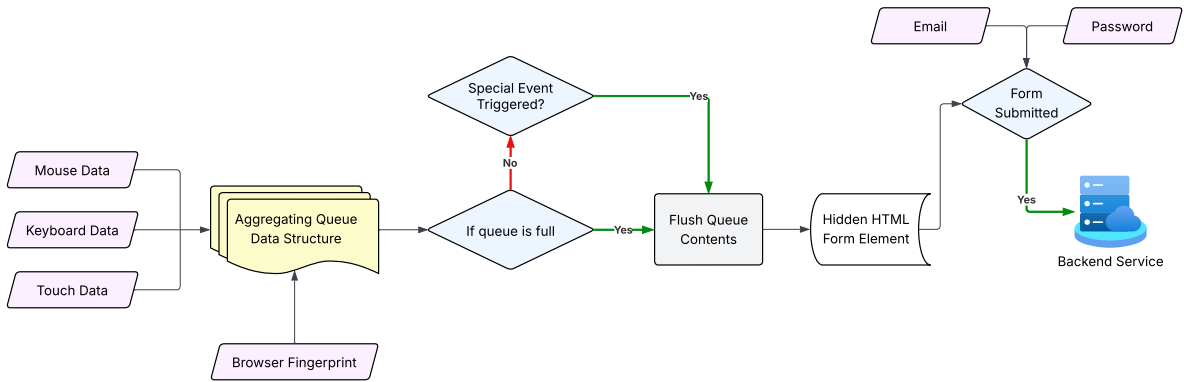


Figure 2.1: NuDetect Collection Flow

Mastercard’s NuDetect was originally a Vancouver-based startup called NuData Security, founded in 2008 (acquired by Mastercard in 2017). The company provided fraud detection services by using hundreds of signals (device, location, keystroke patterns, etc.) to distinguish legitimate users from fraudsters in real-time, without adding friction to the user experience. This system has been widely adopted by major banks and top e-commerce providers to prevent major and costly threats like account takeovers, credential stuffing attacks, and fraudulent transactions [109].

Figure 2.1 shows the flow of behavioral and fingerprinting data in the NuDetect script. This information is collected after manually analyzing the flow of data inside the script using Chrome DevTools [63]. We use the debugger provided by DevTools to put in breakpoints to see the script’s performance in real-time. In Figure 2.1, we show that mouse keyboard and touch data are collected and aggregated to a queue data structure, i.e., whenever a user moves their mouse or types on the keyboard, an additional entry is added to this queue. Simultaneously, data about the users’ browser fingerprints is also appended to this queue, which includes Navigator and Canvas fingerprinting. This data is then encoded into text and placed in a hidden form field on the page HTML (this form field often has the identifier `nds.pmd`). This is done under two conditions: if the queue has been filled, or if some special event has been triggered. Special events include mouse clicks, form submissions, etc., which result in an immediate flush of data to the HTML element. Finally, the data is sent to the backend service (this may belong to NuData or the client website) to detect fraudulent signals after the form has been submitted.

2.4.2 TransUnion Iovation

Iovation, Inc. is a fraud prevention company founded in 2004 [129]. Early efforts by the company were focused on trying to identify users’ devices via accurate fingerprints, including information about hardware, IP addresses, etc. [107]. TransUnion acquired Iovation in 2018 to leverage its large repository of device fingerprints and advanced fraud prevention solutions [159].

Iovation, like NuDetect, collects and aggregates behavioral data into a queue data structure. However, it differs by exposing this data to other JavaScript files on the webpage through appending an object to the global `Window` object. Their object is called `IGL00` and contains a method called `getBlackBox`. This method returns an encoded string containing behavioral data collected over a fixed duration. Essentially, one script gathers behavioral data and makes it available to other scripts via the `Window.IGL00.getBlackBox()` method, allowing this data to be used for risk-based decisions about user identity.

2.4.3 BioCatch

BioCatch is a cybersecurity company that specializes in fraud prevention using behavioral biometrics. It was founded in Tel Aviv in 2011. They used advanced machine learning techniques to analyze behavioral patterns like typing patterns, mouse movements, and

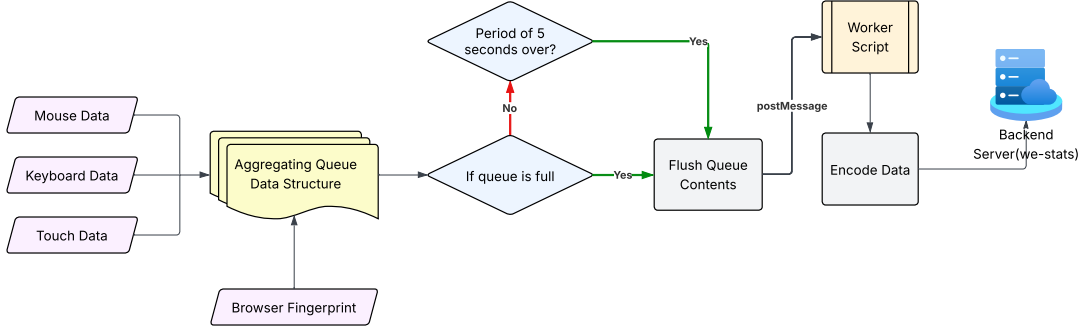


Figure 2.2: BioCatch Collection Flow

touch patterns to create a unique behavioral profile for their users. Their approach allows for continuous authentication throughout a user’s session, improving security without adding friction to the user experience [16]. In May 2024, BioCatch was acquired by Permira Growth Opportunities for approximately \$1.3 billion. BioCatch is one of the leading providers of behavioral biometric-based fraud prevention solutions for banks across the globe, including more than 30 of the top 100 banks [122]. BioCatch acquired a company called AimBrain, a company that provided behavioral biometric solutions in 2020 to improve its solution [13]. BioCatch has also partnered with Microsoft to help secure the Microsoft Cloud for Financial Services (FSI Cloud), to provide anti-fraud support using behavioral biometrics [14].

We first noticed BioCatch on the website of the Bank of Montreal (BMO). BioCatch has a unique way of sending behavioral data to backend servers for processing. As shown in Figure 2.2, the BioCatch script collects behavioral and fingerprinting features and stores them in an in-memory queue data structure. This queue is flushed in two conditions: either the queue is full or a period of 5 seconds has passed; the queue data is then sent to another worker script via the browser’s built-in `postMessage` API. Worker scripts are special scripts that are used to offload heavy work and run in a separate thread from the main browser thread (BioCatch is the only company that we found to use worker scripts for dispatching data over the network). In the case of BioCatch, the worker script uses its `onmessage` handler to accept data from the main collection script. It then converts this behavioral data into a large unsigned 8-bit integer array, which is later converted into a string (an unsigned 8-bit integer can be considered a character). This string is then sent over the network. The network endpoint is sometimes the domain `we-stats.com`, a domain owned by BioCatch. Other times, it may go to a subdomain of the client’s webpage, e.g.,

abc.com/auth/data.

We noticed that in this script, PII (personally identifiable information) such as what key was pressed on the keyboard was not exfiltrated. All keystrokes that are pressed are replaced by the key ‘A’ on the client side. What is sent over the network is mouse movements and timing information for keystrokes, as well as touch coordinates. It may be possible for an attacker to use inter-key latency to infer what keys were pressed, but to the best of our knowledge, no paper directly attempts to execute this attack.

2.4.4 LexisNexis BehavioSec

BehavioSec is a cybersecurity company that specializes in behavioral biometrics-based continuous authentication. It was founded in 2008 in Sweden [164]. Their products augment digital security by doing behavioral analysis, i.e., analyzing behavioral patterns, such as typing rhythms, mouse movements, and touchscreen behaviors. BehavioSec’s fraud prevention solutions have been integrated with platforms like Kount [92]. LexisNexis Risk Solutions acquired BehavioSec in May 2022 [100]. This acquisition was meant to augment their security platform, ThreatMetrix, which offers clients enhanced security measures using an advanced fraud detection framework [134].

The first instance we found of this script was on the Royal Bank of Canada’s website. The script was found using the tool NerdyData [116] and the keyword **behavioSec**. The script operates similarly to the scripts mentioned above; it collects and aggregates behavioral data as well as fingerprinting data. This script, however, has a unique way of dispatching data to the network. Other scripts collect and dispatch data via POST requests, which are considered more secure. BehavioSec dispatches behavioral data periodically via GET requests. The GET request is initiated when the encoded behavioral data is put in the source element for an HTML script or image tag via JavaScript. Listing 2.1 shows a highly simplified implementation of BehavioSec’s code. In the `transmitViaGet` method, adding a `src` to an HTML element triggers an automatic GET request for that URL to optimally retrieve data from that URL.

2.4.5 Ping Identity PingOne

Ping Identity was established in 2002 in Denver, Colorado. They provide identity and access management solutions (IAM) to a wide variety of customers. The PingOne Cloud Platform is one of its flagship offerings. It delivers a comprehensive suite of services designed to secure and protect digital interactions across a variety of applications [126]. In

Listing 2.1: Simplified BehavioSec implementation showing data collection via event listeners (lines 7-22), encoding (lines 27-33), and transmission via GET requests using script/image tags (lines 34-41).

```

1  class BehavioralAnalytics {
2    constructor() {
3      this.eventBuffer = [];
4      this.sessionData = {startTime: Date.now(), interactions: 0, patterns: {}};
5      this.initEventListeners();
6    }
7    initEventListeners() {
8      document.addEventListener('mousemove', (e) => {
9        this.captureEvent('mouse', {x: e.clientX, y: e.clientY,
10          timestamp: Date.now()});
11      });
12      document.addEventListener('keydown', (e) => {
13        this.captureEvent('keyboard', {key: e.key, keyCode: e.keyCode,
14          timestamp: Date.now()});
15      });
16      document.querySelectorAll('input').forEach(input => {
17        input.addEventListener('focus', (e) => {
18          this.captureEvent('form', {field: e.target.name,
19            action: 'focus', timestamp: Date.now()});
20        });
21      });
22    }
23    captureEvent(type, data) {
24      this.eventBuffer.push({type, data});
25      this.sessionData.interactions++;
26      if (this.eventBuffer.length >= 50) { this.transmitData(); }
27    }
28    encodeData() {
29      const encodedData = btoa(JSON.stringify({
30        events: this.eventBuffer, session: this.sessionData,
31        timestamp: Date.now()
32      }));
33      return encodedData;
34    }
35    transmitViaGet() {
36      const encodedData = this.encodeData();
37      const script = document.createElement('script');
38      script.src = `https://analytics.example.com/collect?data=${encodedData}`;
39      document.head.appendChild(script);
40      const img = new Image();
41      img.src = `https://analytics.example.com/pixel?data=${encodedData}`;
42      this.eventBuffer = [];
43    }
44    transmitData() {
45      if (this.config.useSecurePost) { this.transmitViaPost(); }
46      else { this.transmitViaGet(); }
47    }
48  }
49  document.addEventListener('DOMContentLoaded', () => {
50    const analytics = new BehavioralAnalytics();
51    setInterval(() => {
52      if (analytics.eventBuffer.length > 0) { analytics.transmitData(); }
53    }, 5000);
54  });

```

2021, PingIdentity acquired Secured Touch [127], an Israeli startup specializing in behavioral biometrics for fraud detection [125]. PingOne supports the detection of fraud and account takeover attacks using behavioral biometrics, specifically, keystroke dynamics, mouse movements, and touch patterns. They use this in conjunction with their existing fraud prevention network to aid and detect malicious actors detect and prevent fraud early.

The first instance of this script we found was on the login page of the e-commerce website Victoria’s Secret [165]. It has a similar pattern to existing scripts, which collect and aggregate data before sending it over the network periodically. This company, interestingly, provides integrations in their fraud prevention platform, PingFederate, with other popular behavioral biometric providers like LexisNexis’ BehavioSec [124].

2.4.6 D4T4 Celebrus

Celebrus Technologies plc, founded in 1999 and headquartered in the UK, is a digital identity and data platform specializing in real-time, first-party data capture and behavioral biometrics [24]. The company was acquired by D4T4 Solutions in 2015 [32]. The goal of our manual analysis was to identify behavioral biometric companies and try to differentiate them from analytic companies, as they both collect browser fingerprints and behavioral data. For all other companies in our list, we found them to be focusing on fraud prevention using behavioral biometrics. Celebrus is the only company we came across that claims to provide solutions for both behavioral biometrics for fraud prevention and web analytics [25]. While we did find an instance of a Celebrus script on the Hang Seng Bank login page, we were unable to confidently classify it as doing behavioral biometrics or web analytics. So we did not include this Celebrus script for our evaluation.

2.4.7 Forter

Forter is another prominent fraud prevention company founded in 2013. The company offers frictionless automated real-time fraud prevention and detection solutions. These solutions range from account takeover protection to payment fraud prevention. The Forter Trust Platform encompasses various products, including Fraud Management, Payment Optimization, Chargeback Recovery, Account Protection, and Abuse Prevention. These solutions are built upon three core components: the Identity Graph, Network Effects, and the Decision Engine [57]. We found many Forter scripts across the web and most commonly on e-commerce webpages. We noticed especially that these scripts existed throughout the webpage, unlike some other scripts for companies like Mastercard’s NuData, which

primarily exist on login pages. Behavioral biometrics on payment pages may help avoid credit card fraud.

Transmit Security

Transmit Security offers identity and access management solutions, including behavioral biometrics for fraud detection. Their documentation reveals two versions of their behavioral biometric scripts, with the newer version collecting more comprehensive behavioral and fingerprinting data. These enhancements aim to improve the accuracy and reliability of their authentication processes. In our analysis, we found both versions of the script actively recording data on different webpages. We use both versions for our evaluations.

Feedzai

Feedzai [55] is a data science company that develops real-time machine learning tools to identify fraudulent payment transactions. While primarily focused on fraud detection, Feedzai has incorporated behavioral biometrics into its solutions to enhance its capabilities. We found scripts for Feedzai using the Tracker Radar Collector [39]. The script was loaded from the domain `feedzai.cloud`, which belongs to Feedzai.

2.4.8 Bot Detectors

While the majority of companies and scripts we found focus on using behavioral biometrics for preventing account takeovers and credential stuffing, we identified a small subset of companies that use behavioral biometrics for bot detection. As shown in Table 2.2, DataDome [36], ArkoseLabs [6], and Human [76] provide bot detection solutions. Each of these companies has multiple solutions, some of which may rely primarily on browser fingerprints. We found multiple scripts for each company (some collect much more behavioral information than others). Thus, using our best effort methodology, we associate the script that collects more behavioral information (e.g., mouse movements, touch data, and keystroke information) with a behavioral biometric solution for each company. For example, if a company has two scripts, one that collects where clicks happen when you use your mouse (a single feature), versus a script that collects more information (keystroke timings, mouse movement patterns, etc.), we associate the latter with a behavioral biometric solution.

2.4.9 Other Companies

These are some other companies we found interesting in our search for companies that utilize behavioral biometrics. They have unique use cases for behavioral biometrics or may be companies that used the service in the past and no longer do.

Irdeto Devuno Unbotify

Unbotify was an Israeli startup specializing in behavioral biometrics to detect bots and prevent cheating, particularly in gaming. Their technology analyzed user interactions, such as keystroke dynamics and mouse movements, to distinguish between human users and automated scripts. It was acquired by Irdeto's Devuno and is now known as Devuno Unbotify [82]. The interesting thing about this company is that they are targeting a separate market using behavioral biometrics to prevent bots and hacking in video games, showing the breadth of application of behavioral biometrics in the modern digital landscape.

Prove

Prove is a digital identity authentication company that acquired UnifyID in 2021 to enhance its mobile-based passive authentication and behavioral biometrics capabilities [133]. UnifyID's technology utilizes various behavioral patterns, including gait analysis, to authenticate users seamlessly. However, according to our discussion with a Prove representative, gait analysis is no longer employed for authentication purposes.

TypingDNA

TypingDNA specializes in typing biometrics, offering solutions like Verify 2FA, which provides two-factor authentication based on typing patterns, eliminating the need for additional devices. Their ActiveLock product offers continuous endpoint authentication by monitoring typing behaviors to prevent unauthorized access [160]. TypingDNA has 2FA integration with Ping Identity, Okta and Microsoft Entra ID [161]. The most interesting thing about this solution is the second factor authentication, where TypingDNA gives you a string challenge to type in, and using your behavior it verifies the user.

SamsungSDS Nexsign

Samsung SDS’s Nexsign is a FIDO-certified biometric authentication platform that combines various modalities, including behavioral biometrics, to provide secure and seamless user experiences. By analyzing user interactions, such as typing patterns and touch behaviors, Nexsign continuously authenticates users during sessions, reducing the risk of fraud [22]. They have recently integrated with BioCatch’s behavioral biometric solution to enhance their capabilities [139].

Plurilock

Plurilock offers behavioral biometric solutions like DEFEND and ADAPT, which provide continuous authentication by analyzing user behaviors, including keystroke dynamics and mouse movements [128]. These solutions are designed for on-premise deployment, catering to organizations with stringent security requirements. Plurilock’s technologies aim to support zero-trust architectures by ensuring that only authorized users can access sensitive systems. We spoke to a senior executive from Plurilock, who informed us that currently, they offer only on-premise solutions. Since our study focuses on companies that provide products for web-based clients, Plurilock was also excluded from our study.

Graboxy — Cursor Insights

Graboxy, in collaboration with Cursor Insight, focuses on behavioral biometrics by monitoring cursor movements and typing patterns to create unique user profiles [67]. While no active scripts were found during our study, a senior executive confirmed that their technology is used for continuous authentication and competes with companies like Plurilock, BioCatch, and LexisNexis in the behavioral biometrics space.

Authenware

AuthenWare was a company that provided behavioral biometric services, focusing on keystroke dynamics for user authentication. However, the company appears to have ceased operations. Using the Wayback Machine [78], we accessed their website from 2014, which confirmed their offerings in behavioral biometrics. Since we do not find any script from this company, and to the best of our knowledge, the company no longer exists, we label the script found column as N/A in Table 2.1.

Zighra and VeridiumID

Zighra and VeridiumID are providers of behavioral biometric solutions primarily for mobile devices. Zighra’s platform offers continuous authentication by analyzing user behaviors and device interactions, while VeridiumID focuses on multi-factor authentication using behavioral biometrics. Since our study concentrates on web-based clients, these mobile-centric solutions were not included in our analysis.

Table 2.1 and Table 2.2 mention many other companies that we confirmed are using behavioral biometrics in some capacity. We segment the table into 3 sections to segment solutions for account fraud and payment fraud prevention, from companies that mainly focus on bot detection or other companies like Plurilock, TypingDNA, and Unbotify that provide authentication services, but they are distinct from many of the other companies we have mentioned above. We noticed many of these companies have a similar pattern of collecting fingerprinting and behavioral data and aggregating it into some data structure before sending it over the network.

Figure 2.3 from Experian shows a comparison of some companies that do account takeover fraud prevention. This mentions many companies from our list that use behavioral biometrics, among other companies like Entrust that only use browser fingerprints. The figure also attempts to identify vendors that provide fraud-focused services versus vendors focused on authentication. The dark circles represent vendors using a hybrid approach involving both methods.

2.5 Conclusion

In this chapter, we identified, collected, and classified behavioral biometric scripts from a diverse range of companies operating in the fraud prevention and bot detection spaces. Employing a best-effort methodology, we initially established a comprehensive understanding of what constitutes behavioral biometric technologies and subsequently pinpointed providers utilizing keystrokes, mouse movements, and touch dynamics as key biometric signals. Our strategy started from a known industry player (Mastercard NuData) and iteratively expanded to include a total of 39 companies through detailed exploration of vendor documentation, whitepapers, and direct consultation with industry representatives.

We successfully identified and documented scripts on 232 websites associated with major behavioral biometric providers such as BioCatch, BehavioSec, TransUnion’s Iovation, and Forter, among others. Notably, each script employs distinct techniques for data collection,

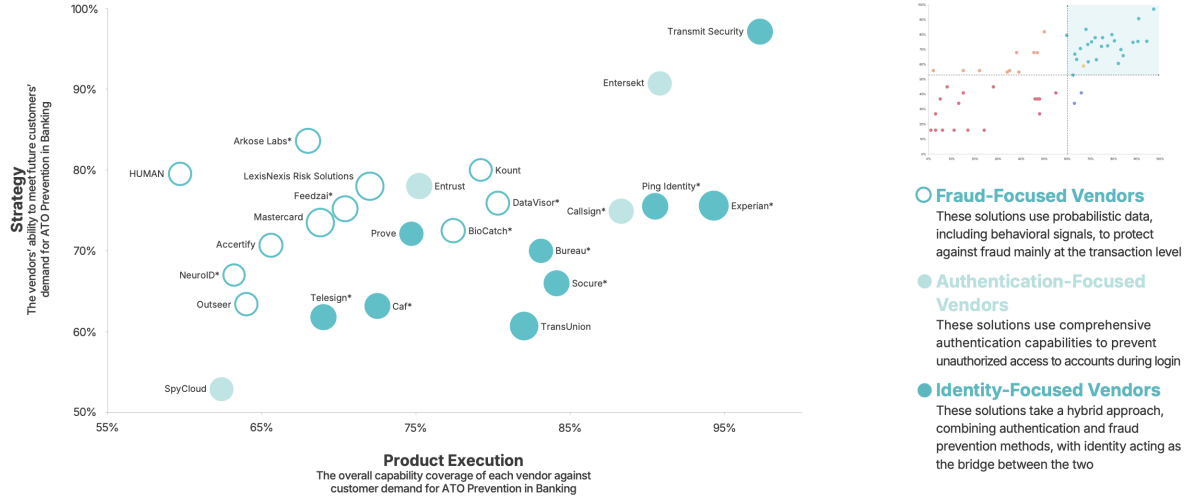


Figure 2.3: This image published by Experian [48], shows the landscape of fraud prevention vendors and their capabilities [47]. Note that many, but not all, of these companies provide behavioral biometric services for fraud prevention.

aggregation, encoding, and transmission, despite sharing common foundational behaviors. For example, NuDetect employs hidden HTML elements to stage data prior to form submission, whereas BioCatch leverages browser worker scripts and message-passing APIs. These nuances highlight the sophistication and subtle diversity in the implementation strategies of behavioral biometric solutions.

Our work provides the first systematic catalog of active behavioral biometric scripts on the web, contributing critical ground-truth data that enables subsequent analytical and evaluative research in this domain. This groundwork also emphasizes the complexity inherent in distinguishing behavioral biometric scripts from general analytics or session-replay scripts, a challenge evident in our analysis of companies like D4T4 Celebris. Moreover, we noted the significant overlap in market segments, with certain vendors serving dual roles in authentication and fraud prevention, reflecting broader industry trends toward hybridized security solutions.

Ultimately, our comprehensive approach and rigorous validation criteria ensure a reliable foundation for the upcoming chapters of this thesis. Moving forward, we will utilize the ground-truth dataset established here to assess the prevalence of behavioral biometric scripts on the modern web [8].

Company	Script Found	Script Active
Fraud Prevention		
BioCatch	Yes	Yes
BehavioSec	Yes	Yes
ThreatMark	Yes	Yes
Mastercard Inc (NuData)	Yes	Yes
Forter	Yes	Yes
Accertify	Yes	Yes
Sardine.ai	Yes	Yes
Transunion (Iovation)	Yes	Yes
Experian (Neuro-id)	Yes	Yes
Bureau.id	Yes	No
Socure	Yes	No
F5 Risk Services	Yes	Yes
Prove Identity Inc	N/A	N/A
PingIdentity	Yes	Yes
Transmit Security	Yes	Yes
Zighra	N/A	N/A
Veridium	N/A	N/A
FeedZai	Yes	Yes
DataVisor	Yes	Yes
ThreatFabric	No	No
SecureAuth Corp	No	No
Authenware	N/A	N/A
CallSign Inc	Yes	Yes
Outseer	No	No

Table 2.1: Behavioral Biometric and Fraud Prevention Companies by Category

Company	Script Found	Script Active
Fraud Prevention (continued)		
Daon	No	No
D4T4 Celebrus	Unconfirmed	Yes
GroupIB	Yes	Yes
Graboxy	No	No
Untarget.ai	Yes	Yes
Yofi	Yes	Yes
Darwinium	Yes	Yes
Cheq	Yes	Yes
MangoPay	No	No
Cleafy	No	No
Deepnet Security	No	No
Bot Detection		
Arkose Labs	Yes	Yes
Human (PerimeterX)	Yes	Yes
DataDome	Yes	Yes
Other		
Plurilock	N/A	N/A
TypingDNA	Yes	Yes
Unbotify	N/A	N/A

Table 2.2: Behavioral Biometric and Fraud Prevention Companies by Category (continued)

Chapter 3

Static Analysis in JavaScript

The following chapter will discuss our approach of using static taint analysis on behavioral biometric scripts to explore distinctive patterns, which will be used for the behavioral biometric script classification model in subsequent chapters.

3.1 Introduction

Behavioral biometric scripts exhibit distinctive patterns of data collection and transmission, as previously discussed in Section 2.4. These scripts gather sensory input such as mouse movements, keyboard interactions, and touch events, subsequently transmitting this data to external network endpoints. Detecting and analyzing these flows systematically is critical to differentiating behavioral biometrics scripts from other scripts. Taint analysis enables the tracing of data from sensitive sources such as browser APIs to their eventual sinks. Existing approaches typically utilize either dynamic or static taint analysis methods to analyze browser fingerprinting [90, 81, 19, 104, 152, 5]. Dynamic analysis suffers from a lack of coverage, if certain functionalities are not triggered on a webpage the code will never be dynamically executed. However, purely static methods also have limitations since they rely solely on code inspection without confirming the actual invocation of sensitive APIs. Our methodological approach enhances traditional static analysis by first dynamically triggering interactions with the webpage using Puppeteer [64] and VisibleV8 [86] and then statically analyzing data flow using Program Dependency Graphs (PDGs). This hybrid approach ensures we analyze only actively invoked APIs during real user interactions, significantly reducing false positives. VisibleV8 captures detailed logs

indicating which APIs are triggered and their locations within the source code. To perform static taint analysis, we construct Program Dependency Graphs (PDGs) [56] using JStap [51], an esprima-based AST generator, extended with control flow and data flow edges. We specifically focus on identifying aggregation points—nodes where multiple data sources converge before being dispatched externally. By quantifying how frequently behavioral and fingerprinting APIs pass through these nodes, our technique highlights areas of high sensitivity and potential privacy risks.

This chapter presents a detailed framework for extracting meaningful features from behavioral biometric scripts through taint analysis. Our hybrid static-dynamic approach identifies key characteristics of data flow patterns and API usage that serve as discriminative features for subsequent machine learning classification of behavioral biometric scripts

3.2 Related Work

Browser fingerprinting detection has used various JavaScript analysis techniques, each with different advantages and limitations that inform our hybrid approach to behavioral biometric script detection.

3.2.1 Static Analysis for Browser Fingerprinting Detection

Zhao [175] developed FProbe, a flow-focused detection framework that performs static data flow analysis on JavaScript code. FProbe focuses on identifying browser fingerprinting by analyzing how different browser attributes are collected and used within scripts, achieving 97.81% accuracy on fingerprinting scripts and 96.31% on real-world JavaScript code from 988,220 websites.

Acar et al. [2] introduced FPDetective, one of the early frameworks for detecting browser fingerprinting through static analysis. FPDetective focuses on font detection practices and uses instrumented browsers to detect fingerprinting scripts without relying on known fingerprinter blacklists. Their large-scale analysis of the top million websites revealed that fingerprinting adoption was much higher than previously estimated.

Haanen et al. [71] proposed using static JavaScript code classification with machine learning to detect browser fingerprinting. Their approach uses Support Vector Machine classifiers to analyze static code features and achieved 94% accuracy in detecting fingerprinting websites. This work demonstrated that static analysis combined with machine learning can effectively distinguish fingerprinting scripts from benign code.

FaizKhademi et al. [50] developed FPGuard, which uses static analysis of predefined metrics to identify fingerprinting attempts. FPGuard analyzes JavaScript code patterns and API usage to detect various fingerprinting techniques including Canvas and Flash-based fingerprinting.

Su and Kapravelos [152] introduced BFAD (Browser Fingerprinting Automatic Discovery), which uses static taint analysis to discover new fingerprinting techniques. BFAD builds program dependency graphs and traces data flows from fingerprinting sources to network sinks without running the scripts. Our work directly extends BFAD’s implementation, adapting their approach for behavioral biometric detection while modifying their requirement that both sources and sinks must be triggered. We found BFAD to be the most relevant static taint analysis work, combining dynamic triggering of browser APIs with static analysis to minimize false positive flows.

Static analysis provides complete code coverage but struggles with obfuscated code and may analyze dead code paths that never execute in practice, leading to false positives.

3.2.2 Dynamic Analysis for Browser Fingerprinting Detection

Iqbal et al. [81] developed FP-Inspector, which combines static code analysis with dynamic taint analysis to detect fingerprinting behaviors. Their machine learning approach analyzes both code structure and runtime behavior, achieving 26% better detection rates than previous methods.

Li et al. [104] introduced FPFlow, which uses dynamic taint analysis to trace fingerprinting data flows during script execution. FPFlow monitors the complete fingerprinting process from data collection to network transmission, achieving high accuracy by eliminating false positives from dead code analysis.

Boussaha et al. [19] presented FP-tracer, which uses dynamic taint tracking and entropy-based thresholds for detailed fingerprinting detection. Their approach extends Foxhound browser with enhanced taint tracking capabilities, supporting 62 sources and 25 sinks to detect various fingerprinting techniques including audio, canvas, and storage fingerprinting.

Sjösten et al. [147] proposed EssentialFP, which uses dynamic information flow tracking to detect fingerprinting by identifying patterns where scripts gather information from multiple browser APIs and communicate it to network sinks. EssentialFP leverages JSFlow, a dynamic information flow tracker for JavaScript deployed in a browser.

Recent work by Bara [11] applies dynamic taint analysis specifically to behavioral biometric script detection, achieving high precision but revealing significant coverage limitations

when behavioral data collection paths remain untriggered during analysis.

The main limitation of pure dynamic analysis is the challenge of systematically triggering all relevant user interactions, device conditions, or temporal events that activate relevant code paths, resulting in a lack of coverage.

3.2.3 Our Hybrid Approach

Our approach addresses the complementary limitations of static and dynamic analysis by using VisibleV8 [86] to dynamically confirm API usage, then applying static analysis to trace all possible flows from these confirmed sources. This methodology eliminates false positives from analyzing unused code while achieving comprehensive coverage of actual data flows.

We leverage VisibleV8’s engine-level instrumentation, which cannot be easily detected or circumvented unlike traditional prototype patching approaches. This provides reliable API monitoring for evasive scripts common in behavioral biometric applications.

Our aggregation analysis combines insights from FPFlow’s dynamic aggregation detection with static flow analysis inspired by BFAD’s taint tracking to identify sophisticated behavioral data collection patterns. This hybrid methodology enables detection of both known and previously unknown behavioral biometric providers while building on the established foundation of fingerprinting detection research.

3.3 Design

This section will detail our static taint analysis system design, along with details about the instrumented browser we used.

3.3.1 VisibleV8

VisibleV8 is an instrumented version of Chrome that addresses critical limitations of traditional JavaScript instrumentation by modifying the V8 JavaScript engine [86]. It aims to improve on in-band methods (also called prototype patching), which is essentially wrapping a native function with a wrapper. Listing 3.1 shows an example of prototype patching. The native add event listener method is replaced by a wrapper that adds additional logging for the method. Listing 3.2 shows how to detect prototype patching. All native methods, such

as `addEventListener` return the string `'[native code]'`, so if a function is modified, adversarial code can use this to know if that is the case. VisibleV8 gets around this by putting its instrumentation directly in Chrome’s V8 engine. Their approach intercepts all native function and property accesses, including access to fingerprinting APIs and global objects. Along with detailed information on what browser APIs are triggered, the system also tells what part of the JavaScript code is triggering each API. VisibleV8 also comes with a highly scalable crawling framework [168], allowing the usage of this system in large-scale studies.

Listing 3.1: Monkey-patching technique showing function interception (line 6) and instrumentation injection (lines 7-9) for event listener tracking

```

1 (function() {
2   // Store original function
3   const originalAddEventListener = EventTarget.prototype.addEventListener;
4
5   // Replace with instrumented version
6   EventTarget.prototype.addEventListener = function(type, listener, options) {
7     if (type === 'mousemove' || type === 'click') {
8       console.log('API Access:', type, 'listener registered');
9     }
10    return originalAddEventListener.call(this, type, listener, options);
11  };
12 })();

```

It is important to note that VisibleV8 (VV8) can only report browser APIs for a script if that script actively uses them. For example, when a user moves their mouse on a website, all scripts can potentially access mouse movement events, but only scripts that specifically register to listen for these events using `addEventListener` will actually be detected by VV8 as accessing mouse APIs. Listing 3.5 shows an example where a script subscribes to the `click` event (i.e., each time a click occurs, this code will be triggered). This method has access to all properties of the `MouseEvent` class (we call these properties the browser APIs). VisibleV8 will only collect the specific APIs that the script uses (e.g. `MouseEvent.screenX`, `MouseEvent.isTrusted`, etc.); it will not report other APIs accessible via `MouseEvent` but not used within a particular script. For each API, it reports that it was collected as well as the character offset of the respective script where the API was triggered (e.g., {476, `PointerEvent.screenX`}). We use these character offsets later in Section 3.3.3 to search for specific nodes in our PDG.

As mentioned in the previous section, each of these APIs needs to be *collected*, but as mentioned, the code in Listing 3.5 is only triggered when a user interacts with the webpage. So, behaviors must be triggered before they can be collected. For this, we modify the crawler code in VisibleV8’s crawler. We use Puppeteer to simulate normal user behavior and trigger all APIs that may be relevant to user behavior. Listing 3.3 shows a high-level implementation of our crawler’s functionality. The crawler visits a page, waits for it to load,

and then triggers a variety of behaviors using Puppeteer’s browser automation interfaces, rather than programmatic JavaScript events (e.g., `document.body.click()`). We do this as we found that behavioral biometric scripts would often check the `isTrusted` property for each event to see if this was triggered via JavaScript or an actual user interaction. Puppeteer’s interface is ideal to circumvent this check since it uses the Chrome DevTools Protocol to simulate real mouse movements, clicks, and keyboard events that appear as genuine user interactions to the browser.

Listing 3.2: Anti-instrumentation detection showing native code verification (lines 4-5) and conditional execution based on detection results (lines 8-13).

```

1 // Real-world evasion technique found in behavioral biometric scripts
2 function isInstrumented() {
3   // Check if addEventListener has been monkey-patched
4   const nativeCode = EventTarget.prototype.addEventListener.toString();
5   return 'nativeCode.includes('[nativeCode]');
6 }
7 // Behavioral biometric script logic
8 if (isInstrumented()) {
9   // Detected instrumentation - alter behavior or exit silently
10  return;
11 } else {
12   // Safe to collect behavioral data
13   document.addEventListener('mousemove', collectMouseData);
14 }

```

This behavioral data will eventually leave the webpage via sinks. We define a sink as the last place data exists in JavaScript code, before being sent to the network, browser storage, HTML, etc. Often, behavioral biometric scripts periodically send data over the network, meaning these sinks are automatically triggered and collected by VV8. However, some scripts will send data on form submission to the network. Triggering form submission for modern websites can be challenging due to a variety of website structures and varying navigation flows. Hence, our crawler does not attempt to trigger form submission sinks.

3.3.2 PDG & JStap

Given APIs triggered by each script on a webpage using VisibleV8, we now want to perform taint analysis from behavioral and fingerprinting sources to relevant sinks. For such an analysis, we require some sort of program dependence graph (PDG). The idea for such a graph was originally introduced by Ferrante et al. [56]. It provides a unified representation capturing both control and data dependencies within a program’s source code. A PDG can be defined as a directed graph $G = (N, E)$ where:

- Nodes N represent statements or expressions from the source code.

Listing 3.3: Behavioral tracking detection crawler showing browser initialization (lines 4-6), event triggering sequence (lines 12-19), and data collection phase (lines 22-28).

```

1  async function webCrawler(url, uid) {
2    // Initialize browser with stealth plugins and HAR recording
3    const browser = await puppeteer.launch({headless: true, stealth: true});
4    const page = await browser.newPage();
5    const har = new PuppeteerHar(page);
6
7    // Navigate to target URL
8    await page.goto(url, {waitFor: 'load'});
9    await sleep(4000);
10
11   // Trigger behavioral events
12   await triggerEventHandlers(page);
13   await triggerClickEvents(page);
14   await triggerFocusBlurEvents(page);
15   await triggerMouseEvents(page);
16   await triggerKeyboardEvents(page);
17   await triggerScrollEvents(page);
18   await triggerTouchEvents(page);
19   await triggerWindowResize(page);
20
21   // Wait for events to be propagated
22   await sleep(loiterTime * 1000);
23
24   // Save results and cleanup
25   await har.stop();
26   fs.writeFileSync(`${uid}.har`, harData);
27   await browser.close();
28   await cleanupTempFiles();
29 }

```

- Edges E signify dependencies, classified into:
 - **Data dependency edges**, indicating that the definition of a variable at one node is utilized at another.
 - **Control dependency edges**, which reflect conditional execution; a node n_j is control-dependent on node n_i if the execution of n_j depends on the outcome of node n_i .

These graphs are especially useful for their capacity to precisely track how data propagates across different control flow paths, enabling highly granular analyses [73].

JStap [51] is a static analysis framework designed specifically for pre-filtering potentially malicious JavaScript code. It constructs static representations of JavaScript code, aiming to efficiently detect malicious scripts without executing them. The tool first generates an Abstract Syntax Tree (AST) using Esprima [89], it then subsequently enhances this AST into a Control Flow Graph (CFG) and Program Dependence Graph (PDG). It does so by adding additional control and data flow edges to the existing AST.

Listing 3.4 shows a simple Javascript program, we use this to illustrate a simple Program Dependence Graph (shown in Figure 3.1) (we adopt the same example used by the authors

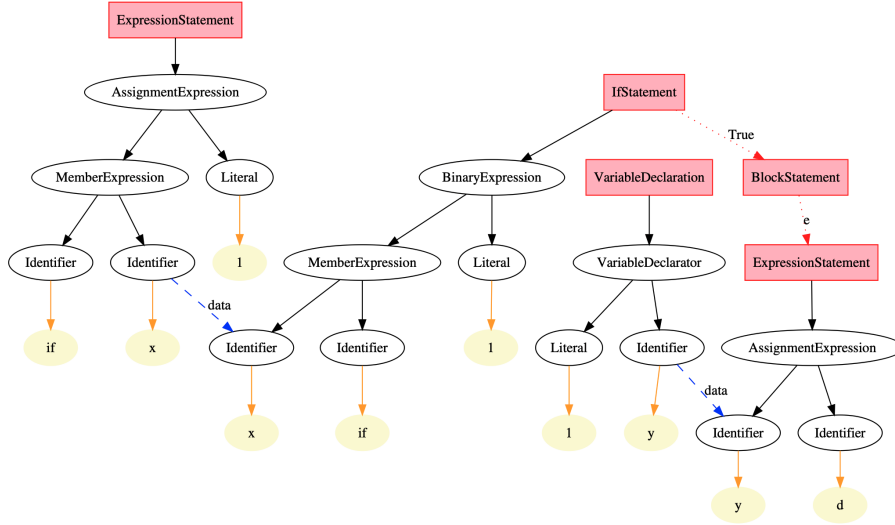


Figure 3.1: Simple PDG for the program in Listing 3.4

of JStap in their paper [51]). The PDG representation captures both the structural and dependency relationships within the code. At the top level, we see three main statement nodes (shown in red rectangles): an **ExpressionStatement** for the assignment `x.if = 1`, a **VariableDeclaration** for `var y = 1`, and an **IfStatement** for the conditional logic. Each statement is decomposed into its constituent expressions and operators, represented as oval nodes in the graph.

The data dependency relationships are particularly important for taint analysis. For example, the blue “data” edge connecting the identifier `x` from the first statement to the

Listing 3.4: Simple JavaScript variable assignment and conditional check example

```

1 x.if = 1;
2 var y = 1;
3 if (x.if == 1) { d = y; }

```

`MemberExpression x.if` in the conditional statement shows how the value assigned to `x.if` flows to its usage in the if condition. Similarly, another data dependency edge connects the variable `y` to its usage in the assignment `d = y` within the conditional block. Control dependencies show execution relationships: the “True” edge from the `IfStatement` to the `BlockStatement` indicates conditional execution, while “e” edges represent unconditional sequential flow between statements where execution always proceeds regardless of any conditions.

This explicit representation of both data and control flows enables precise tracking of how values propagate through the program, which is essential for our taint analysis to trace behavioral biometric data from API sources to potential exfiltration sinks.

In our work, we leverage JStep to systematically generate PDGs for JavaScript code. By representing data flows explicitly through PDGs, we can accurately trace sensitive data propagation from browser API sources to sinks.

Listing 3.5: Simple data collection and aggregation example in JavaScript

```

1 document.addEventListener('click', clickHandler);
2 function clickHandler(e) {
3     let attr1 = e.screenX;
4     let attr2 = e.screenY;
5     let attr3 = e.isTrusted;
6     let attr4 = e.target;
7     let attr5 = e.currentTarget;
8     let inputElement = document.querySelector('input');
9     let oldValue = inputElement.value;
10    if (attr5 == 62) {return;}
11    else {inputElement.value = oldValue + attr1 + attr2 + attr3 + attr4 + attr5;}
12 }

```

3.3.3 Taint Analysis

While pure dynamic analysis often struggles to identify all data flows from sources to sinks, as not every code path can be triggered during web interactions, purely static analysis can result in a large number of false positives by analyzing flows that may never execute in practice. To address the limitations of both approaches, we propose a hybrid approach that combines the strengths of both methods (dynamic analysis for collection and static taint analysis for data propagation).

Hybrid Methodology

Our hybrid methodology is grounded in the observation that behavioral biometric scripts tend to initialize and trigger behavioral data collection APIs early during webpage loading,

especially on login pages, due to their security-focused purposes. Dynamically capturing script behavior is a reliable way to learn what APIs each script is collecting while ignoring irrelevant APIs. Unlike FP-Inspector [80], which statically examines API invocations by keyword search, we dynamically confirm the invocation of the relevant API, significantly reducing false positives by limiting our analysis to paths involving APIs genuinely executed at runtime. We then analyze the static paths for source data, from these APIs, until it reaches a sink. This, in contrast with dynamic taint analysis, can report data flows that may not have been triggered during runtime. Scripts that send their data over the network upon form submission are difficult to detect dynamically, since, as mentioned in Section 3.3.1, it is challenging to trigger form submissions at scale during web crawling. In such cases, static taint analysis can detect these data flows without requiring them to be dynamically triggered.

For our analysis, sources consist of known browser fingerprinting APIs from existing research [80, 151, 19], as well as key behavioral APIs. A full list is mentioned in Appendix C.1.1. We consider all behavioral APIs that inherit from the `UIEvent` class in Chrome. We also put our list of sinks in Appendix C.1.3.

We record triggered API events dynamically using VisibleV8 (VV8). It provides detailed logs on precisely which APIs were triggered, their invocation frequency, and specific offsets within the source code, recording this information into a structured PostgreSQL database for subsequent static analysis.

Graph Construction with JStap and PDGs

Our static analysis phase leverages JStap [51], which employs Esprima-generated Abstract Syntax Trees (ASTs) enhanced with Control Flow Graphs (CFGs) and Program Dependence Graphs (PDGs). PDGs explicitly represent data and control dependencies, facilitating a robust analysis of data propagation pathways (as mentioned in Section 3.3.2).

However, we encountered challenges constructing graphs due to limitations in the Node.js Esprima package, particularly its inability to handle newer JavaScript syntax like the rest and spread operators (`...`). Additionally, obfuscation techniques frequently employed by behavioral scripts posed significant hurdles (some obfuscation techniques add additional non-functional code to modify script aesthetics, increasing file size), leading to graph construction timeouts and occasional AST generation failures. Notably, scripts associated with Google Tag Manager consistently experienced graph generation timeouts, prompting manual exclusions to maintain overall analysis efficiency.

To mitigate these issues, we increased timeout thresholds, introduced parallel graph construction, and employed telemetry to monitor failures. Despite these optimizations, occasional graph construction failures persisted, primarily with large scripts that take too long for graph construction. We also replaced the `esprima` node package with `esprez` [173], which helps handle errors related to newer syntax.

Flow Analysis and Aggregation Strategy

For our static taint analysis, we modify the implementation of BFAD [151], an existing system that uses locality to find new fingerprinting APIs. Their approach only investigates a script if both a sink and a source API have been triggered by that script; otherwise, the script is discarded. However, since we may not be able to trigger all sinks (form submission, etc.) on a page, we analyze all scripts that collect behavioral and fingerprinting data from our list of source APIs. We trace whether these data flows reach any of the triggered sinks, and if they do not, we extract other features such as data aggregation patterns. We use a modified list of sinks and sources as mentioned in Section 3.3.3.

Our analysis incorporates an aggregation concept inspired by FPFlow [104], addressing limitations inherent in tracing obfuscated or incomplete source data paths. We constructed Program Dependence Graphs (PDGs) to precisely identify nodes where multiple sensitive APIs converge, effectively pinpointing aggregation points within scripts. By counting unique API occurrences traversing these nodes, we measure their relative importance for behavioral and fingerprinting data flows.

Explicitly, we perform API name matching on PDG nodes—for instance, searching for `userAgent`—to identify data flows connecting triggered sources to confirmed sinks. For each node along a path, we record the distinct APIs that traversed it, calculating aggregate scores separately for fingerprinting APIs, behavioral APIs, and their combined total. This approach allows us to extract a wide variety of features for scripts through API aggregation. We use this as a feature since we noticed many behavioral biometric scripts aggregate behavioral and fingerprinting features before sending them to a sink.

Compared to purely static methods, our hybrid approach offers several important advantages. By dynamically confirming which APIs are genuinely triggered during user interactions, we significantly reduce false-positive results associated with purely static analyses. This practical confirmation ensures our static analysis focuses solely on relevant code paths, enhancing both accuracy and reliability. Additionally, purely dynamic analyses often miss deeply nested or rarely activated code paths, making them incomplete. By incorporating

static analysis after dynamically verifying API invocations, we ensure better coverage, even for complex or infrequent code paths.

Our real-world analysis further revealed that behavioral biometric scripts consistently use a wider range of fingerprinting and behavioral APIs compared to other script categories, like session replay and analytics scripts. These biometric scripts often aggregate multiple types of user interactions—mouse movements, keystrokes, and touch events—to create detailed behavioral profiles. Our hybrid method accurately captured these nuanced patterns by analyzing actual API invocation frequencies. This method enables us to precisely extract features from scripts on the web to help with the classification of behavioral biometric scripts.

3.3.4 System Design and Scalability

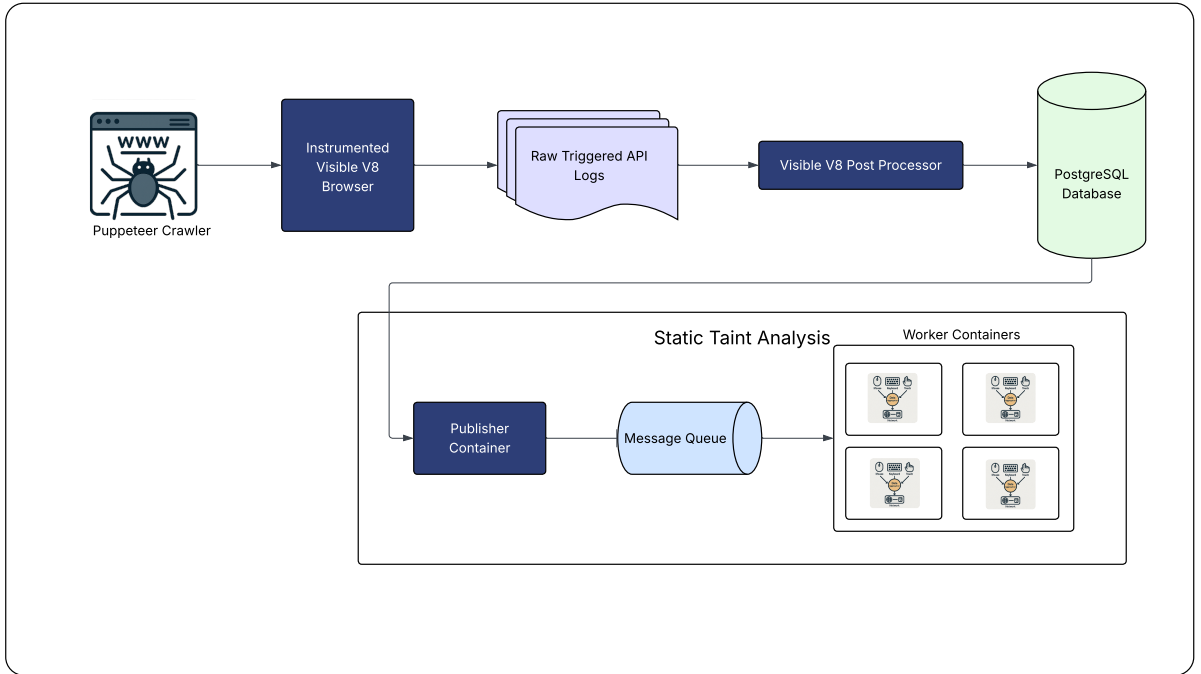


Figure 3.2: Taint Analysis System Diagram

Figure 3.2 presents the overall architecture of our static analysis system, designed explicitly to handle the computational demands associated with large-scale web studies. Constructing detailed data flow graphs can be time-consuming, often taking up to seven minutes

per graph due to necessary timeout constraints. We chose seven minutes as our threshold after testing on an initial set of approximately 2,200 scripts (described in Section 5.2), where more than 90% successfully generated graphs within this timeframe. We are able to successfully construct graphs for 21 of the 23 vendors that we have found scripts for. Graph construction for CallSign and F5 Risk Services fails due to timeouts (these scripts are very large in size). Increasing the threshold to 9 or even 10 minutes did not improve graph construction success rates. Recognizing this computational challenge, we prioritized scalability through parallelization and modular design.

To achieve efficient parallel processing, we isolate the graph construction and static taint analysis tasks for each JavaScript file within independent Docker worker containers. This design ensures multiple analyses can run concurrently, significantly improving throughput and effectively managing large datasets (hundreds of thousands of script files).

The analysis pipeline, as illustrated in Figure 3.2, begins with a Puppeteer-based crawler that dynamically triggers browser APIs within the VisibleV8 (vv8) instrumented browser. The VV8 browser logs all API interactions at runtime. These logs are processed by vv8’s post-processing module, which associates triggered APIs with their corresponding scripts and stores this structured information in a PostgreSQL database. A dedicated publisher container continuously monitors the database, identifying new scripts to process and placing corresponding tasks into a message queue. Worker containers subsequently retrieve tasks from this queue, generate Program Dependence Graphs (PDGs) using JStap, and perform comprehensive static taint analysis. The final analysis outcomes and extracted features are stored back in the PostgreSQL database. These features will later be used by our machine learning classifier to differentiate between behavioral biometric and non-behavioral biometric scripts.

3.4 Limitations

While static taint analysis can detect flows that may not be triggered dynamically, it may also encounter issues with dead code (code that exists but can never be triggered). This would result in false-positive flows to sinks and aggregation that can never occur in practice. One way of solving this is by using dynamic taint analysis in conjunction with our static taint analysis to minimize such false positives. The work of Bara [11] uses dynamic taint analysis for this use case; we believe our approach, when combined with theirs, provides a more comprehensive understanding of script behavior. Static taint analysis complements dynamic taint analysis by addressing the inherent limitations of each approach: static

Listing 3.6: Obfuscated simple data collection and aggregation example in JavaScript

```

1 document.addEventListener('click', clickHandler);
2 function clickHandler(e) {function(){
3   var _0xabc1=["\x61\x64\x64\x45\x76\x65\x6E\x74\x4C\x69\x73\x74\x65\x6E\x65\x72","\x63\x6C\x69\x63\x6B","\x71\x75\x
   ↪ \x65\x72\x79\x53\x65\x6C\x65\x63\x74\x6F\x72","\x69\x6E\x70\x75\x74","\x76\x61\x6C\x75\x65","\x73\x63\x72\x
   ↪ \x65\x65\x6E\x58","\x73\x63\x72\x65\x65\x6E\x59","\x69\x73\x54\x72\x75\x73\x74\x65\x64","\x74\x61\x72\x67\x
   ↪ \x65\x74","\x63\x75\x72\x72\x65\x6E\x74\x54\x61\x72\x67\x65\x74"];
4
5   document[_0xabc1[0]](_0xabc1[1],function(e){
6     var a1=e[_0xabc1[5]];
7     var a2=e[_0xabc1[6]];
8     var a3=e[_0xabc1[7]];
9     var a4=e[_0xabc1[8]];
10    var a5=e[_0xabc1[9]];
11    var e1=document[_0xabc1[2]](_0xabc1[3]);
12    var ov=e1[_0xabc1[4]];
13    if(a5==((-63))) {
14      return;
15    }
16    else {
17      e1[_0xabc1[4]]=ov+a1+a2+a3+a4+a5;
18    }
19  });
20 } }();

```

analysis mitigates the coverage gaps of dynamic analysis, while dynamic analysis reduces the false positives common in static analysis.

In addition to this, we faced graph construction timeouts while attempting to construct graphs for extremely large scripts. One notable example is Google Tag Manager’s script, which always times out for graph construction in our seven-minute construction threshold. It is, however, important to note that graph construction only failed for one of our 19 behavioral biometric companies (CallSign). This is a limitation in the way JStap constructs graphs. One possible solution would be to parallelize graph construction for JStap. We leave such improvements to future work.

While our analysis focuses on data flow within individual JavaScript files, there may also be cases where data is collected and aggregated by one script but transmitted over the network by a different script. This methodology is employed by scripts like BioCatch, as described in Section 2.4.3. To address this, we use the inter-script communication (`postMessage` API) as a sink. However, there are other ways to do cross-script communication, such as attaching an object to the global `Window` object. Our static approach is currently unable to detect such cross-script flows. Hence, we recommend a combination of dynamic and static taint analysis to address this issue as well.

Another main challenge with static taint analysis is code obfuscation. Listing 3.6 shows an obfuscated version of Listing 3.5. While the two codes are functionally equivalent, their appearances vary drastically. The important thing to notice is that the sink in this example is the `HTMLInputElement.value` API which can be seen on line 11 in the unobfuscated Listing 3.5. The same sink is used in line 17 for Listing 3.6; however, the

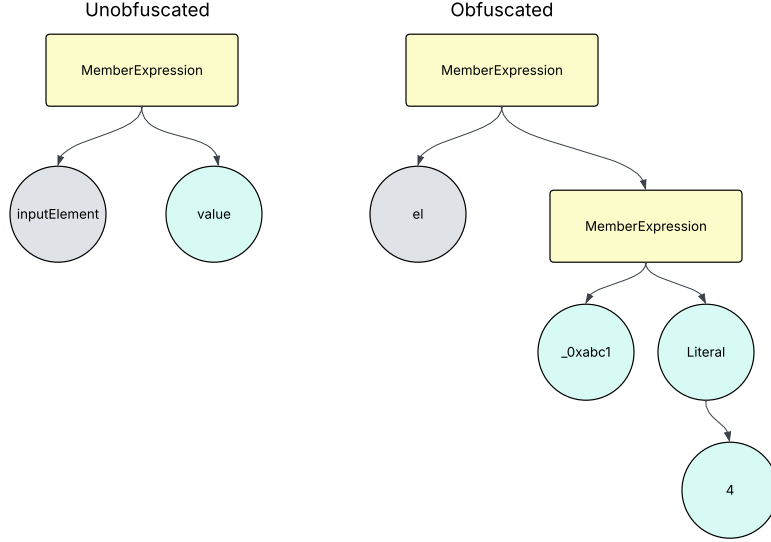


Figure 3.3: Obfuscated AST vs Unobfuscated AST

API **value** cannot be seen directly in the code. VisibleV8 reports that this specific API is triggered at a particular offset in the code, but these offsets are not always perfectly accurate. While they consistently point to the approximate location, we often need to scan the surrounding characters to locate the exact position of the relevant sink API. For example, if VisibleV8 reports an offset that corresponds to `[_0xabc1 [4]]`, we must search the nearby code region to identify the actual API usage. To perform static analysis using JStap, which stores these offsets in the graph nodes, we need to find the graph node that corresponds to this offset and contains the API keyword **value**. However, this approach fails in heavily obfuscated code where API names are completely obscured or dynamically constructed, as there may be no recognizable API identifiers to match against the reported offsets. Figure 3.3 shows the contrast between the different ASTs for both versions of the code; the nodes highlighted in blue refer to our sink (`HTMLInputElement.value`). We cannot search the obfuscated version for the **value** keyword, hence we will be unable to detect a flow to this sink in our analysis. While there has been work on JavaScript deobfuscation [72, 119, 146], the approaches sometimes result in code that does not compile. Also, the deobfuscated code will vary structurally from the code found on the browser, hence the VisibleV8 offsets will not be transferable. Hence, this is a limitation of the static taint analysis approach. The best solution is a union our approach with Bara [11] to use dynamic taint analysis to catch flows that are missed statically.

3.5 Conclusion

In this chapter, we introduced a hybrid static-dynamic taint analysis approach tailored specifically to explore and characterize behavioral biometric scripts. By combining the precision and comprehensiveness of static analysis with the accuracy and reduced false-positive rate offered by dynamic API triggering via VisibleV8, our method effectively navigates the challenges inherent in analyzing JavaScript code at scale. This integration allows us to systematically trace sensitive data from their initial collection points through aggregation nodes and eventually to their final sinks, effectively capturing nuanced data-flow patterns characteristic of behavioral biometric scripts.

However, our methodology is not without limitations. Obfuscation techniques, prevalent across many behavioral scripts, present significant obstacles, often hindering precise static analysis. Additionally, extremely large or highly obfuscated scripts pose challenges in Program Dependency Graph construction, occasionally leading to timeouts or graph generation failures. While strategies such as extending graph construction thresholds and parallel processing significantly mitigate these issues, future improvements could incorporate more advanced deobfuscation or inter-script analysis techniques to provide even more comprehensive coverage.

Despite these limitations, our approach was able to successfully construct PDGs and identify dataflow patterns in scripts from 21 of the 23 companies for whom we have scripts, demonstrating its practical effectiveness. The hybrid static-dynamic taint analysis we developed demonstrates clear advantages over traditional purely dynamic or static methods, particularly by reducing false positives and expanding coverage to include rarely activated code paths. Importantly, our findings illustrate that behavioral biometric scripts uniquely aggregate diverse types of user interaction data—such as mouse, keyboard, and touch events—thus forming distinctive behavioral fingerprints. This chapter, therefore, sets the stage for the subsequent classification model, which leverages these discovered features to effectively differentiate behavioral biometric scripts from other script categories on the web.

Chapter 4

Login Page Crawler

The following chapter will discuss a novel web crawler that finds login pages using Visual-Language Models. This crawler will be used to detect and categorize login pages in the remainder of this thesis.

4.1 Introduction

In the context of websites, login pages can be defined as the entry points to the sensitive part of a web application [162]. Drakonakis et al. [38] classify login pages as pages with a visible password field. Jonker et al. [85] further segment these login pages into two categories: one-step login pages where a password and identifier input field (e.g., username, email, user-id, etc.) is displayed simultaneously, and two-step login forms where first the identifier is requested and then after submission the password form is displayed. In our study, we define a login page as any page where there is an input field relevant to a login form, it may either be a one-step or two-step login form as defined above.

During our initial exploration and discovery of companies that do behavioral biometrics (described in Chapter 2), we noticed that behavioral biometric scripts exist on login pages more frequently than on other web pages. Therefore, we want to be able to access login pages on modern websites at scale. Traditional login page crawlers rely on using heuristics to get to login pages, limiting their adaptability to dynamic page structures and multilingual web applications. Recent systems have attempted login page detection using structured form analysis, but struggle to handle unconventional page designs [123, 144]. Furthermore, the set of heuristics used is not comprehensive enough for multilingual websites [83]. To

address these limitations, we have developed LoginGPT a novel login page crawler that uses Vision Language models to crawl web pages and classify login pages based on visible form fields. Our evaluation shows that LoginGPT outperforms existing systems, finding significantly more login pages, while maintaining high precision.

4.2 Related Work

4.2.1 Heuristics-Based Login Crawlers

Automating login page detection on the web has been an open problem for the research community. Senol et al. [144] use keyword-based crawling combined with a machine learning classifier based on 88 features extracted from the webpage. Their tool is called Double Edged Sword. However, this approach has two major limitations. Firstly, the keywords used are language dependent, and they fail to capture non-text elements like icons that may take you to a login page. Secondly, their machine learning classifier relies on features extracted from the HTML of a page, meaning it can tell if a login form exists on the page HTML but not how to make it visible (e.g., clicking on an element will open a modal showing the login form). As a result, pages with no visible login forms are classified as login pages, with no information on how to make the forms visible. This is especially problematic, as in our case, certain behavioral biometric scripts are only loaded if the login form is visible on the page.

Jonker et al. [85] made a crawler that uses a variety of methods to find login pages, including using search engines as well as a keyword-based crawler with a crawl depth of one (only one level of clicks will be performed on the home page). Although this approach has more potential to uncover a login page, it classifies these pages based on keywords. These keywords may not extend to other types of websites such as banking websites, where instead of login a user might see 'Online Banking' and clicking this would take them to a login page. Furthermore, with a click depth of one, the system is not able to handle the complicated login flows in modern websites.

Most recently, SSO-Monitor by Jannet et al. [83] uses multiple strategies to crawl and detect login pages including search engines, crawling (depth two), sitemaps and common login paths to detect login pages. They use these methods to propose login page candidates with varying priorities, later these candidates are filtered based on heuristics. For example, using the LastPass [97] extension in Chrome. LastPass is a password manager that can be used to detect login elements on a webpage. SSO-Monitor uses this detection to classify

a login page candidate as a true login page. While this is a useful heuristic, we found that the extension suffers from high false negatives and mainly focuses on pages where both the identifier and password fields exist.

The flaws of these systems lead us to design LoginGPT, a heuristics-free system that classifies pages based on visible page elements using Vision Language Models.

4.2.2 Vision Language Models for Web

Recent advances in large language models (LLMs), more specifically, vision language models (VLMs) have allowed better navigation and testing of user interfaces (UIs) for modern web pages. Commercial VLMs (e.g., OpenAI, Claude and Deepseek) have promising performance on general vision-based tasks, but are not particularly reliable when it comes to image grounding (i.e., finding coordinates of elements in an image) [99, 167, 170].

Vision models are leveraged to understand GUI screenshots without relying solely on the HTML DOM. Cheng et.al introduced SeeClick [27], a vision GUI-agent that can perform image grounding using only image data. They achieve this by pre-training their model on a large set of GUI grounding data achieving high accuracy on the Screenspot benchmark, introduced within the paper. Building on this, Wu et al. developed OS-ATLAS [170], a foundation model for GUI agents aimed to create a robust image grounding model. The model is trained on a large open-source GUI dataset with more than 13 million elements for Windows, Mac, Linux, Android and Web UIs. OS-ATLAS fine tunes Alibaba’s Qwen2-VL [166], an open-source VLM with strong image text recognition, compared to other open-source models like LLaVa [4] and Flamingo [105]. OS-ATLAS achieves state-of-the-art GUI grounding accuracy on multiple benchmarks significantly outperforming other open-source models.

Visual GUI agents like UGround [66] take a similar approach by training their universal GUI-grounding model using extensive synthetic data, giving a 20% accuracy gain over methods like SeeClick. AGUVIS [171] adds to this by creating a general-purpose UI agent that can plan and execute multi-step high-level tasks, using image grounding. Collectively, these works show the efficacy of multimodal vision models for GUI based tasks, such as recognizing and isolating elements, such as forms, buttons, banners, etc.

For our case, this approach is ideal as we find that many behavioral biometric scripts are only loaded onto the page when a login form is visible. We hypothesized that vision-based models may perform better than heuristics for identifying elements that may lead to a login page. We chose OS-ATLAS specifically for its state-of-the-art open-source performance,

robust training on web UI data, and strong image grounding capabilities that are essential for accurately identifying login-related elements across diverse website layouts.

In the following section, we describe the design of our system and compare the systems performance with existing login page crawlers SSO-Monitor and Double Edged Sword.

4.3 LoginGPT

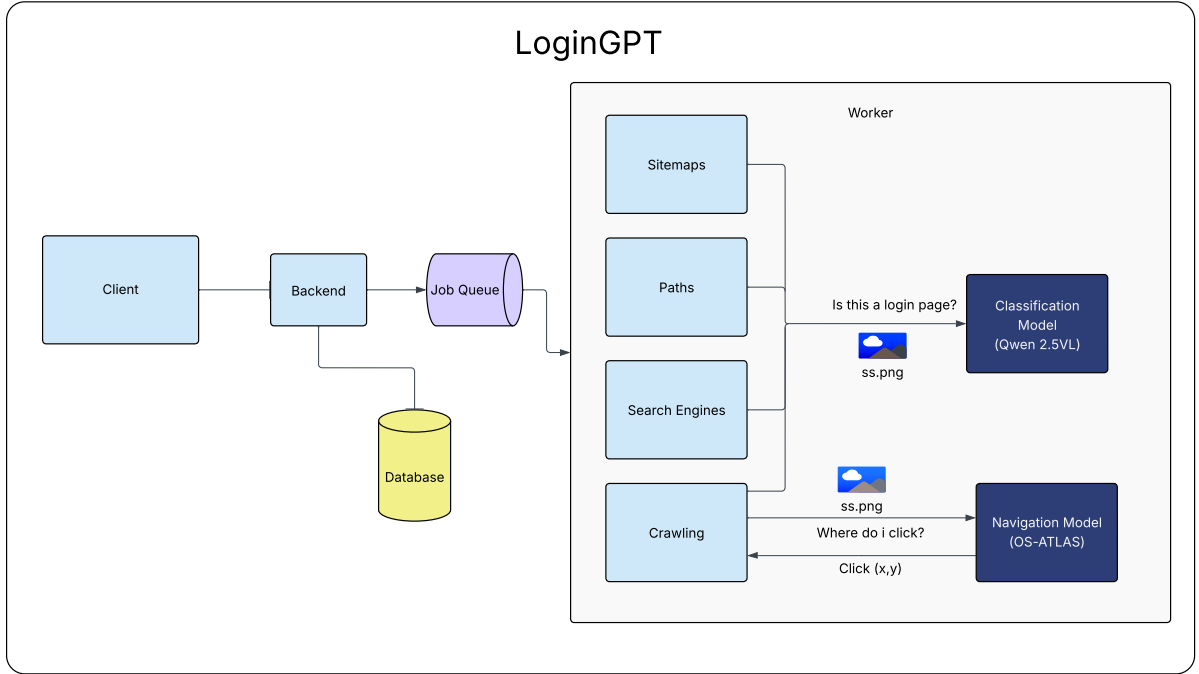


Figure 4.1: LoginGPT system diagram

To overcome the challenges faced by existing heuristics-based login crawlers, and use the versatility of vision language models, we introduce **LoginGPT**, a crawler that utilizes multimodal inference to interact with and classify web pages based on their visible elements, rather than text-based heuristics. We extend the existing codebase for SSO-Monitor [83] since it already provides login page candidates via search engines, sitemaps and common paths. We replace their heuristics based crawling method with our own as described in Section 4.3.2.

4.3.1 Login Page Detection Strategies

Figure 4.1 shows the high-level design of LoginGPT. A client sends a request to find login pages for one or more domains. The system backend receives this request and adds it to a job queue. Multiple workers who are already subscribed to the job queue execute crawls to one website at a time. Each worker attempts to sequentially find login pages via four strategies:

1. **Sitemaps:** A structured list of all pages on a website, it can be considered a roadmap of all possible subpages you can visit for a given domain. We adopt the existing logic of SSO-Monitor, which attempts to find these sitemaps for websites that may show a path to a login page.
2. **Paths:** Common paths used to reach login pages (e.g., `/login`, `/signin`). The system appends these paths to the URL of a given website and checks if this results in a 404 (page not found) error. If a page does not result in this error, it is passed forward into the classification step mentioned below.
3. **Search Engines:** Search for login pages for each website on popular engines like Google, Bing, Yahoo, Brave, etc. This is done using SearXNG [140], a privacy-preserving meta-search engine, which aggregates results from different search engines.
4. **Crawling:** Our crawler, discussed in Section 4.3.2, uses Vision Language models to crawl login pages, allowing it to work for multiple languages and varying page structures across the web, without relying on a limited set of heuristics.

To classify login pages, each strategy takes a screenshot of candidate login pages and sends it to the Classification Model as shown in the figure, to see if it is a login page with a visible login form or not. The model response is binary: Yes/No. Upon collection of all login pages, the result is then sent back to the backend, which then stores them in a NoSQL database.

4.3.2 LLM Crawler

To avoid the issues with existing heuristics-based crawlers mentioned in Section 4.2.1, we propose our own crawler, which uses Vision Language models to traverse websites, in order to discover login pages. As shown in Figure 4.2 the system goes to a click depth of five. We do this in agreement with previous work to make crawling feasible on a scale and to

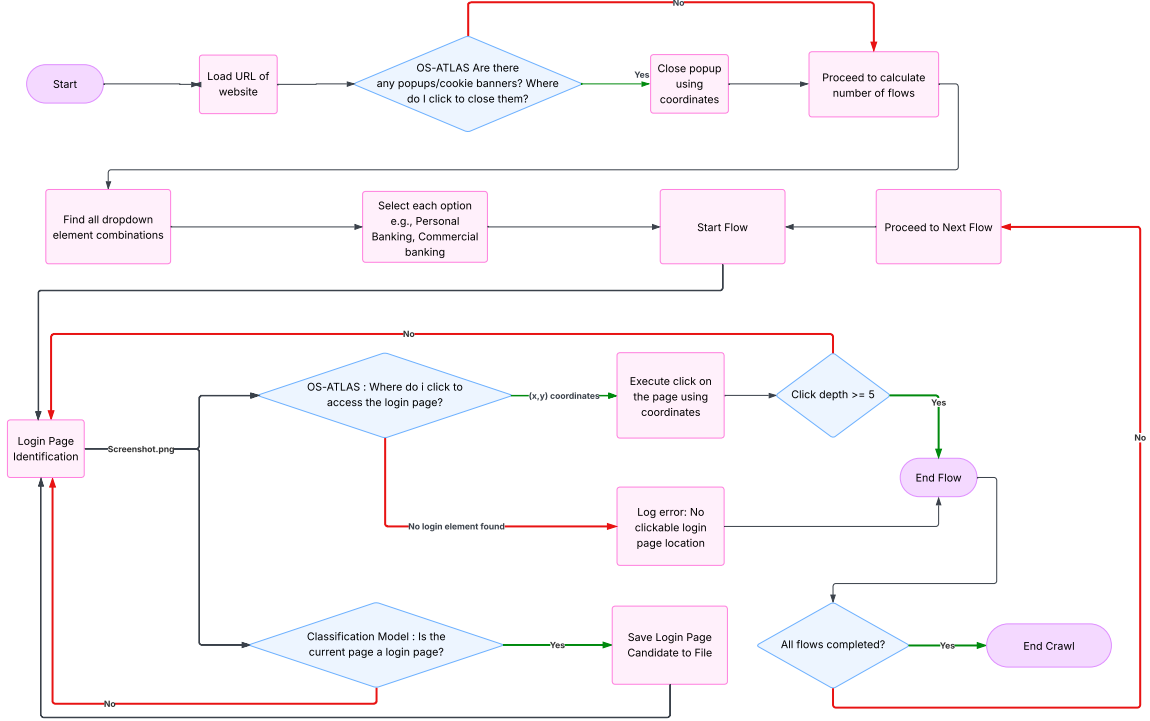


Figure 4.2: Flowchart showing crawler’s internal logic

avoid our crawler getting stuck in bot traps [29] or in logical loops (cyclically repeating the same flows in a website).

We manually visited the top 100 websites from the CrUX list [41], and 100 banking websites in the U.S., to find common patterns among login flows. These banking websites are a collection of two lists, one is an open-source list of banks in the US, CommunityBanker [154] and the other is the FDIC-insured banks in the US [52]. We noticed that the existing crawlers [83, 144] performed poorly, especially on banking websites. The reason is, that the focus of heuristics-based crawlers was on finding keywords like **login**, **signin** etc; however, banking websites contain keywords such as **Online Banking**, **Personal Banking** etc. In addition to this, on normal websites, we observed similar issues where we often found icons instead of login buttons, and a variety of login flows. We also noticed that many pages have a popup such as a cookie banner or notification/promotion. In many cases, these popups can only be closed by clicking on an **Accept** button, or **Close** button. Finally, banking websites can have multiple login pages, one for Personal Banking, Business

Banking, Loans, etc. We noticed that to get to different login pages for the 100 banking websites, one must select which type of banking they want to access. Often this selection was done via a dropdown on the page to select the type of banking.

To cater to these variations, we designed our crawler using Puppeteer [64], on top of the existing implementation in SSO-Monitor, as shown in Figure 4.2. The crawler visits a given URL, finds all dropdown options that can be selected on the page, and for each selection, it starts a flow (e.g., select Personal Banking, Business Banking, Loan). For every flow it attempts to find all login pages within that flow using OS-ATLAS and our Classification Model.

For each crawl to a URL:

- The crawler waits for the homepage to load (i.e. until the load event is fired).
- The crawler takes a screenshot of the page and sends it to the OS-ATLAS (also referred to as the Navigation Model) model via sockets (we chose this model for its quick inference and accurate image grounding).
- First, it asks the Navigation Model where to click to close the popup if it is visible.
- After closing the popup, if found, the system proceeds to find the number of possible login flows using dropdown elements (these may contain different types of accounts, e.g., Commercial Banking, Personal Account, Business Account).
- For each dropdown option on the page, our system attempts to find login pages after selecting this option. We call the usage of each of these options a Webpage Flow.
- For each flow, the crawler takes another screenshot and sends it to OS-ATLAS. OS-ATLAS then attempts to find visible elements on the page to click to access the login page. These may be buttons labeled **Login** or icons that give the affordance of leading to a login page. OS-ATLAS then determines the best location to click to reach a login page and returns this as a response to our crawler (this process is blocking; the crawler waits for the Navigation Model’s response before proceeding).
- Simultaneously, the crawler sends a screenshot of the same page to the Classification Model, which determines if we are at a login page or not (This call is non-blocking as an optimization to speed up crawls).
 - If a login page is found we save this candidate login page to a file along with the actions (clicks) we did to get to this page.

- After the response from the Navigation Model,
 - If the Navigation Model responds that no login element is found, we end our crawl for this specific flow.
 - If the Navigation Model returns a valid response (coordinates), our crawler attempts to click on the provided coordinates. If, after this click, a new tab is launched, we use the new tab as our current tab and continue searching for login elements until the click depth of 5 is reached. We do this since in our manual investigation we found that clicking a login button often resulted in a new tab being opened.
- After the click depth of 5 elements is reached our crawler terminates the flow for this webpage.
- The crawler then checks if there are more flows to be completed, it proceeds to the next flow. If all flows for the webpage have been completed, the crawl for this URL is terminated.

The exact prompts used for our study of the navigation model can be found in Appendix [A.1](#).

The login URL of a website can be extensively long (it may contain details about the browser, the user and other unique identifiers for a particular session) and may be of less use without the path used to find the login page, i.e., what clicks were performed on each page leading upto the login page. We refer to these clicks as actions; we store the login URL as well as the set of actions required to reach a given login page in our NoSQL database. This helps increase the usability of the tools since other researchers can effectively replicate the steps taken to reach the login page from the homepage, in case the login URL is not accessible directly.

Within the crawler we also use Puppeteers Extra Stealth Plugin [12]; it attempts to add a user agent and other necessary features to allow website owners to prevent bot detection. In addition to this, we remove the `navigator.webdriver` property of our browser; this tells the webpage if the browser is being controlled by an automated crawler. We also inject randomized mouse movements to avoid being flagged as a bot by web pages.

We find our approach works better than heuristics in many settings i.e., where different keywords are required to reach a login page (e.g., Personal Banking), and for websites in different languages (e.g., French, Italian, Mandarin, Japanese).

4.3.3 Classification Model

The four strategies mentioned in Section 4.3.1 propose multiple login page candidates. To filter these candidates and find login pages, we use another Vision language model, Qwen 2.5VL. We chose the Qwen model since it is one of the best open-source vision models for image reasoning [174, 26, 103, 84]. As shown in Figure 4.1 each strategy sends a screenshot of their login page candidates to the Qwen 2.5 Classification Model. The model takes the screenshot with a prompt mentioned in Appendix A.1.3. The model then returns an answer containing YES or NO. If the answer is YES, we consider the candidate as a login page.

We use this method of filtering candidates since for our study a login form must be visible on the page for it to be considered a login page. A single login page URL may contain a login form but it may only show after clicking a particular button. This means it is not a login page until the specific button is pressed.

To justify the usage of our system as a login crawler versus existing crawlers like SSO-Monitor and Double-Edged Sword, we compare the performance of our system against these two systems in the following section.

4.4 Evaluation

In this section, we evaluate the performance of LoginGPT versus Double-Edged Sword [144] and SSO-Monitor [83]. We compare the systems on 50 Random Banking websites from the US from our list mentioned in Section 4.3.2 and the top 1000 websites from the CrUX list [41]. We find our system outperforms both Double-Edged Sword and SSO-Monitor in login page detection on the CrUX 1000 websites, finding **67.7%** more unique login pages than SSO-Monitor and **434.8%** more login pages than Double-Edged Sword. More notably, it finds at least one login page on **27.5%** more domains compared to SSO-Monitor and **294.2%** when compared with Double-Edged Sword.

4.4.1 System Configuration

We use for our evaluation of all three systems a machine with 2 AMD EPYC 7272 (2×12 -core processor) with 140GB of system memory, as well as 2 NVIDIA A40 GPUs, each with 48GBs of VRAM. All three systems are run with 25 parallel workers that crawl 25 websites in parallel. In addition to this, we use vllm [93] for hosting our Vision Language Models,

since it allows on-the-fly dynamic batching for inference requests, which is particularly useful for multiple crawls sending inference requests at different times.

4.4.2 Bank Login Pages

Table 4.1: Comparison of Login Page Detection Systems on 50 random US bank websites

Metric	LoginGPT	SSO-Monitor	DES
Candidate Login Pages Found	66	29	12
True Login Pages Found	60	27	9
At Least One True Login Page Found	40	22	9
Precision (%)	90.91	93.1	75.0
Website Coverage (%)	80.00	44.00	18.00

Banks are an important part of our study since banking websites are a core customer for behavioral biometric scripts. Banks may have multiple kinds of accounts, (i.e., Business, Personal, Loans). Each account often has a distinct login page. Our LLM Crawler attempts to explore each page using different options selected on the page as mentioned in Section 4.3.2.

We crawl 50 random websites for banks in the US with all three systems and the results of the crawl can be seen in Table 4.1. From the 50 websites, three crawls resulted in failure: two failed due to a 403 error and one failed due to a timeout when attempting to visit the domain. In the remaining 47 successful crawls our system, LoginGPT, found 66 login pages. We manually checked each page and found that 60 of these were true login pages. We define a true login page as a page where a login form is visible. In addition to this, the system finds at least one true login page for 40 out of the 50 domains (i.e., 80% of domains). This compared with SSO-Monitor, which finds 29 candidate login pages, out of which 27 were true login pages. In addition to this, it found at least one true login page for 22 of the 50 domains. Finally, DoubleEdgedSword found 12 candidate login pages out of which 9 were true login pages. It found at least one login page for 9 domains (since it only proposes one login page per domain).

Our system finds at least one login page on 81.8% more domains than SSO-Monitor and 444.0% more domains than DoubleEdgedSword. While manually labeling these pages, we found that the increased success of our system can be attributed to the replacement of heuristics with Vision Models. The precision (i.e., if a page classified as a login page is a

true login page or not) of SSO-Monitor is highest since it uses the LastPass extension [97] for classifying login pages, as opposed to our system that uses another Vision Model to classify screenshots of pages. Since the latter is more probabilistic in nature, the precision of our system is slightly less than SSO-Monitor but we argue that it is still an acceptable figure that can be used for our study.

4.4.3 CrUX Top 1000

Table 4.2: Summary of Domain Accessibility in 1000 CrUX Websites

Status Category	Count	Percentage (%)
Successful (Reachable)	900	90.0
Access Denied/Restricted	51	5.1
Timeout Issues	23	2.3
Resource Not Found	12	1.2
Connection Issues	10	1.0
Other Server Errors	4	0.4
Total	1000	100.0

We also evaluate our crawler against SSO-Monitor and Double-Edged Sword on the top 1000 websites, in accordance with previous work [144]. For our crawl of 1000 websites, there were 100 failures as mentioned in Table 4.2 (a more detailed breakdown can be found in Appendix A.2). There are 51 Access Denied (often 403 errors), which may be the result of IP blocking since our machine may have been flagged as an automated crawler for many websites that share bot detection providers. The second most prominent error was timeouts (23 domains failed). We checked these timeouts manually and 10 of these timeouts were due to the domain being unreachable, 13 were due to our crawler timing out while navigating to these pages. In addition to this, there were 19 errors due to the page not existing (ERR_NAME_NOT_RESOLVED and Resource Not Found errors).

As shown in Table 4.3, of the 900 successfully crawled websites, LoginGPT successfully found at least one true login page on 612 domains. This is 27.5% more domains than SSO-Monitor and 294.2% more domains than Double-Edged Sword. In addition to this, it found 887 unique login pages, 67.7% more than SSO-Monitor and 426.4% more than Double-Edged Sword. We manually checked all candidate login pages and found our system proposed a total of 1,982 candidate login pages out of which 874 were non-unique

Table 4.3: Login Page Detection Performance by System

System	Domains w/ Login	Unique	Non-Unique	Invalid	Precision
LoginGPT	612 (61.2%)	887	874	221	88.8%
SSO-Monitor	480 (48.0%)	529	333	114	88.3%
DoubleEdgedSword	208 (20.8%)	208	0	15	93.3%

(meaning a page which has a slightly different URL, e.g, `abc.com/login/return?` versus `abc.com/login/`) and 221 were invalid. The precision of the system is calculated as follows:

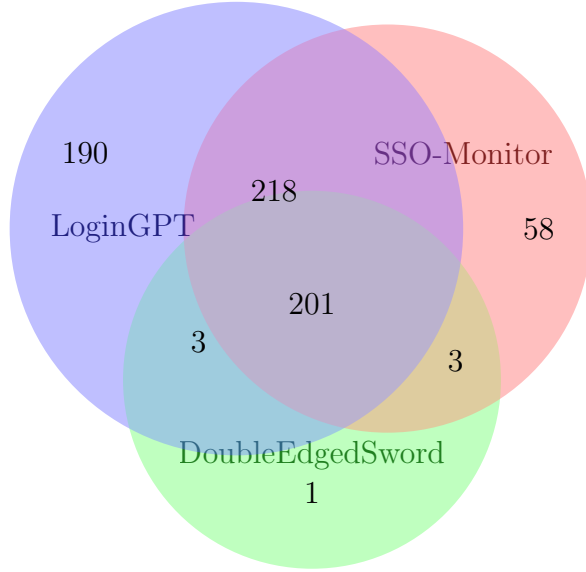
$$\text{Precision} = \frac{\text{True Login Pages}}{\text{Total Candidate Login Pages Detected}} = \frac{\text{Unique} + \text{Non-Unique}}{\text{Unique} + \text{Non-Unique} + \text{Invalid}}$$

Our system has a precision of 88.8%, which is slightly higher than 88.3% for SSO-Monitor. The highest precision, however, was for DoubleEdgedSword, which uses several features to classify login pages (form fields, URLs, HTML elements etc.). However, it proposes fewer login pages than our system. It may be worth consideration for future work, to use a combination of Double-Edged Sword’s, SSO-Monitor’s and LoginGPT’s login page classification strategies to maximize our systems precision.

We analyze the domains where each system found login pages and the overlap between the systems as shown in Figure 4.3. All three systems find login pages for 201 domains, while LoginGPT and SSO-Monitor find login pages for 419 domains. There are 62 domains where LoginGPT failed to find a single login page; 61 were found by SSO-Monitor and 4 by DoubleEdgedSword. We manually checked these domains and found that it was due to our classification method, which misclassified some login pages. This can be addressed by supplementing our classification method with that from SSO-Monitor and DoubleEdgedSword, reducing missed login pages. However, we note that our system exclusively found true login pages for 190 domains, which SSO-Monitor and DoubleEdgedSword missed.

We also analyze the efficacy of each login page detection strategy, mentioned in Section 4.3.1, as shown in Figure 4.4. Our LLM Crawler finds the majority of login pages, with 43.3% of login pages found. Paths and Search Engines find 30.1% and 25.9% pages, respectively. Finally, Sitemaps found only 0.7% of the login pages. This emphasizes our LLM crawlers’ capability for finding login pages on modern websites.

Overall, we find our system finds login pages for at least 27.5% more domains than SSO-Monitor and DoubleEdgeSword on the CrUX top 1000 websites. It also finds at least 67.7% more unique login pages than the existing systems. This shows the advantages of using Vision Models for web navigation over traditional heuristics-based approaches.



Total Domains Analyzed: 1000 (Found Login: 674, No Login: 324)

Figure 4.3: Domain Coverage Overlap Between Login Page Detection Systems

4.5 Conclusion

In this chapter, we presented LoginGPT, a novel vision-based login page crawler that uses Vision Language Models to find login pages across diverse modern web pages. Our approach combats the limitations of existing heuristic-based login crawlers by focusing on visible elements on the page, using accurate UI grounding models. In our comprehensive evaluation, we find that LoginGPT outperforms existing login crawlers SSO-Monitor and DoubleEdgedSword in terms of finding unique login pages, by 67.7% and 426.4%, respectively, while maintaining a high precision rate. We attribute this increase to using Vision Language models trained on a large set of labelled web data, which can operate in multilingual settings and execute complex login flows for modern websites. By successfully detecting login pages for 61.2% of the top 1000 websites, LoginGPT establishes a solid foundation for future web crawlers and navigation agents that use Vision Models to make navigation decisions in today’s complex digital landscape.

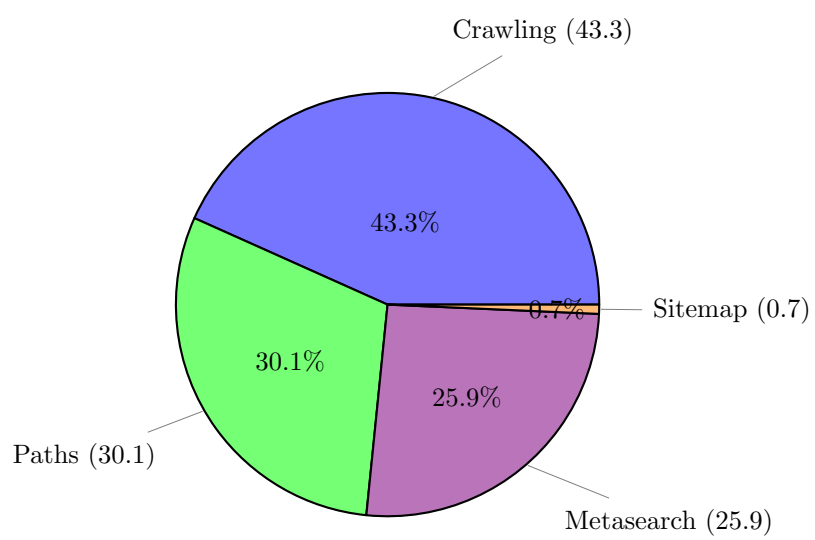


Figure 4.4: Distribution of LoginGPT detection strategies across analyzed domains.

Chapter 5

Behavioral Biometric Script Classification

5.1 Introduction

Behavioral biometrics are commonly employed to enhance security for sensitive workflows on the web. We have observed examples of such behavioral biometric scripts embedded in financial institutions and e-commerce websites. These scripts typically collect behavioral data (e.g., mouse movements, keyboard typing patterns, touch interactions) as well as fingerprinting data (e.g., user-agent details, browser plugins).

The primary goal of our research is to automatically identify behavioral biometric scripts on the web at scale. Previously, as described in Section [2.3.1](#), we relied on a manual approach to identify vendors providing behavioral biometric services and their associated scripts. This manual process is labor-intensive, time-consuming, and difficult to scale.

To address these challenges, we propose automating the classification of behavioral biometric scripts using supervised machine learning techniques. Our approach has two main objectives:

- Discover new behavioral biometric vendors and their scripts on the web.
- Identify scripts from known vendors described in Chapter [2](#).

We trained and evaluated several supervised machine learning models, including Gradient Boosting, Random Forest, and Naive Bayes classifiers, using a custom dataset manually

constructed from over 200 scripts across 18 distinct behavioral biometric vendors. We compare each model’s effectiveness in distinguishing behavioral biometric scripts from other types of scripts, ultimately selecting the best-performing model for our final evaluation.

5.2 Dataset and Labeling

For this chapter, we manually compiled a list of 232 websites [9] that deploy behavioral biometric service providers, drawn from our comprehensive exploratory study presented in Chapter 2. From these websites, we find 2,229 scripts that collect behavioral data, out of which 232 are for behavioral biometrics. These 232 scripts belong to 18 unique vendors (some vendors have multiple scripts). We manually label the positive and negative classes of scripts using keywords mentioned in Appendix B.

We use VisibleV8 and our static analysis methodology mentioned in Chapter 2 to extract raw features for each of these 232 scripts. Table 5.1 shows all raw features our static analysis extracts from each script. These include features relevant to data flow to aggregation and flow to sinks, along with collection information such as access counts to each source API (`behavioral_apis_access_count`). This helps us discover the diversity of features collected as well as the intensity of data collection for each script in our dataset. We consider aggregation as we noticed a common trend in Chapter 2 where scripts collect behavioral and fingerprinting data and then aggregate it into a queue data structure before, later dispatching it on the network. In the following sections, we will utilize these features to construct our classifier.

Table 5.1: Script Classification Raw Features and Descriptions

Feature	Description
<code>max_api_aggregation_score</code>	Highest aggregation score among all API calls made by the script at any one node
<code>behavioral_api_agg_count</code>	Total number of behavioral APIs that were aggregated together at a single node
<code>fp_api_agg_count</code>	Total number of fingerprinting APIs that were aggregated together at a single node
<code>max_aggregated_apis</code>	Maximum number of APIs (fingerprinting and behavioral) aggregated together in a single node
Continued on next page	

Table 5.1 – continued from previous page

Feature	Description
<code>max_behavioral_api_aggregation_score</code>	Highest aggregation score specifically for behavioral APIs
<code>aggregated_behavioral_apis</code>	List of behavioral APIs that were aggregated during script execution (max at any single node)
<code>max_fingerprinting_api_aggregation_score</code>	Highest aggregation score specifically for fingerprinting APIs
<code>aggregated_fingerprinting_apis</code>	List of fingerprinting APIs that were aggregated during script execution (max at any single node)
<code>fingerprinting_source_apis</code>	Unique Source APIs used by the script for device or browser fingerprinting
<code>behavioral_source_apis</code>	Unique Source APIs related to user behavior tracking and monitoring
<code>behavioral_source_api_count</code>	Total number of unique behavioral source APIs accessed by the script
<code>fingerprinting_source_api_count</code>	Total number of unique fingerprinting source APIs accessed by the script
<code>behavioral_apis_access_count</code>	JSON object mapping behavioral APIs to their access frequency counts
<code>fingerprinting_api_access_count</code>	JSON object mapping fingerprinting APIs to their access frequency counts
<code>dataflow_to_sink</code>	Boolean indicating whether data flows from sources to sensitive sinks
<code>apis_going_to_sink</code>	JSON object detailing which APIs have data flowing to sensitive sinks

5.3 Classification Methodology

This section details our script classification methodology including initial heuristic-based classification and supervised machine learning based classification.

5.3.1 Heuristics Based Classification

Before deploying more sophisticated supervised machine learning models, we first attempted simpler rule-based heuristics to investigate whether behavioral biometric scripts could be reliably distinguished from non-behavioral scripts using straightforward threshold-based rules. This approach relies on manually identified patterns observed frequently in behavioral biometric scripts, such as API usage patterns, event aggregation, and data flows.

Initially, we implemented a set of basic heuristics involving minimal thresholds (the following thresholds are selected after observing a subset of our dataset):

- Behavioral API count must be at least 3.
- Fingerprinting API count must be at least 2.
- Behavioral API aggregation score or total aggregation score must reach a threshold of 5.
- At least one behavioral API must exhibit repeated accesses (at least 5).
- If graph construction succeeded, data flow to a sink was required.
- In cases where graph construction failed, stronger behavioral indicators (aggregation scores above 10 and API counts above 5) were required as a fallback.

The results from these simple heuristics were unsatisfactory, yielding high precision (**94.1%**) but very poor recall (**13.8%**), indicating that while most identified positives were correct, the heuristics missed a large portion of actual behavioral biometric scripts (200 out of 232 missed). This confirmed that behavioral biometric scripts could not be easily separated using overly simplistic rules.

In response, we developed a more complex heuristic framework involving weighted rules and threshold scoring, attempting to better capture nuanced behavioral biometric patterns:

- Assign weighted scores to various features, such as behavioral and fingerprinting API counts, their ratio, aggregation scores, a normalized burst of API calls, presence of specific sinks (e.g., storage writes, beacon usage, script injection), and presence of high-signal events (device motion, touch events, etc.).
- A script was classified as behavioral biometric if its total weighted score surpassed a specific threshold.

The optimized heuristic model significantly was trained on 70% of the dataset and evaluated on the remaining 30% (using vendor-aware splitting). The model had improved recall of **69.3%** but the precision was **48.0%** on the test set. This leads to an unmanageable number of false positives. Additionally, this improvement introduces a critical caveat: the manually tuned weights and thresholds could inherently reflect biases specific to our labeled dataset. Such heuristics might overfit our current dataset, resulting in reduced performance when applied to unseen scripts from different vendors or websites. Thus, although our advanced heuristic approach demonstrates promising results, its practical applicability at scale remains uncertain due to potential overfitting and limited generalization.

To address these limitations, we next explore supervised machine learning approaches that leverage automated feature selection and model optimization, aiming for greater generalization and robust performance beyond our initial labeled dataset.

5.3.2 Feature Engineering and Selection Methodology

To build a robust classifier for identifying behavioral biometric scripts, we attempt to create a comprehensive set of vendor-agnostic features. Our goal was to develop generalized features that reflect script behavior, rather than specific implementations tied to certain vendors.

Feature Engineering

Table 5.2: Complete Set of Vendor-Agnostic Features for Behavioral Biometric Script Detection

Feature Name	Type	Description
AGGREGATION FEATURES		
<code>max_api_aggregation_score</code>	Float	Maximum aggregation score among all APIs accessed by the script. Set to 0 if NaN or -1. Represents the highest level of data aggregation complexity.
<code>behavioral_api_agg_count</code>	Int	Count of behavioral APIs that perform aggregation operations. Set to 0 if NaN or -1. Measures behavioral tracking APIs with data aggregation.
<code>fp_api_agg_count</code>	Int	Count of fingerprinting APIs that perform aggregation operations. Set to 0 if NaN or -1. Measures fingerprinting APIs with data aggregation.
<code>total_aggregation_count</code>	Int	Sum of behavioral and fingerprinting API aggregation counts. Computed as <code>behavioral_api_agg_count + fp_api_agg_count</code> .
<code>has_aggregation</code>	Binary	Binary indicator (0/1) for whether the script performs any API aggregation. Computed as <code>int(max_agg > 0)</code> .

Continued on next page

Table 5.2 – continued from previous page

Feature Name	Type	Description
behavioral_agg_ratio	Float	Ratio of behavioral aggregation to total aggregation count. Computed as <code>behavioral_agg / total_agg</code> if total > 0, else 0.
fp_agg_ratio	Float	Ratio of fingerprinting aggregation to total aggregation count. Computed as <code>fp_agg / total_agg</code> if total > 0, else 0.
has_behavioral_aggregation	Binary	Binary indicator (0/1) for whether the script performs behavioral API aggregation. Computed as <code>int(behavioral_agg > 0)</code> .
has_fp_aggregation	Binary	Binary indicator (0/1) for whether the script performs fingerprinting API aggregation. Computed as <code>int(fp_agg > 0)</code> .
has_both_aggregation_types	Binary	Binary indicator (0/1) for whether the script performs both behavioral and fingerprinting aggregation. Computed as <code>int(behavioral_agg > 0 and fp_agg > 0)</code> .
BEHAVIORAL FOCUS FEATURES		
behavioral_focus_ratio	Float	Ratio of behavioral APIs to total APIs. Computed as <code>total_behavioral / total_apis</code> if total > 0, else 0.
fp_focus_ratio	Float	Ratio of fingerprinting APIs to total APIs. Computed as <code>total_fp / total_apis</code> if total > 0, else 0.
INTERACTION PATTERN FEATURES		
interaction_diversity	Int	Count of different user interaction types monitored (mouse, keyboard, touch, pointer, device, wheel, focus). Computed by analyzing event types in behavioral APIs.
has_multi_input_types	Binary	Binary indicator (0/1) for whether the script monitors 3 or more input types. Computed as <code>int(len(event_types) >= 3)</code> .
SOPHISTICATION FEATURES		
tracks_coordinates	Binary	Binary indicator (0/1) for tracking mouse/pointer coordinates (clientX, clientY, screenX, screenY, pageX, pageY).
tracks_timing	Binary	Binary indicator (0/1) for capturing timing information (timestamp, interval).
tracks_device_motion	Binary	Binary indicator (0/1) for accessing device motion or orientation APIs.
sophistication_score	Int	Sum of coordinate, timing, and device motion tracking capabilities. Computed as <code>tracks_coordinates + tracks_timing + tracks_device_motion</code> .
FINGERPRINTING CATEGORY FEATURES		
uses_navigator_fp	Binary	Binary indicator (0/1) for whether the script uses Navigator APIs for fingerprinting. Detected by checking for 'Navigator.' in fingerprinting APIs.
uses_screen_fp	Binary	Binary indicator (0/1) for whether the script uses Screen APIs for fingerprinting. Detected by checking for 'Screen.' in fingerprinting APIs.
uses_canvas_fp	Binary	Binary indicator (0/1) for whether the script uses Canvas or WebGL APIs for fingerprinting. Detected by checking for 'Canvas' or 'WebGL'.
uses_audio_fp	Binary	Binary indicator (0/1) for whether the script uses Audio APIs for fingerprinting. Detected by checking for 'Audio' in fingerprinting APIs.
fp_approach_diversity	Int	Sum of different fingerprinting approaches used. Computed as <code>uses_navigator_fp + uses_screen_fp + uses_canvas_fp + uses_audio_fp</code> .
ACCESS INTENSITY FEATURES		
collection_intensity	Float	Total API access count divided by number of unique APIs. Computed as <code>total_accesses / max(total_apis, 1)</code> . Indicates data collection aggressiveness.

Continued on next page

Table 5.2 – continued from previous page

Feature Name	Type	Description
behavioral_access_ratio	Float	Ratio of behavioral API accesses to total accesses. Computed as <code>total_behavioral_accesses / max(total_accesses, 1)</code> if total $\neq$ 0, else 0.
DATA FLOW FEATURES		
has_data_collection	Binary	Binary indicator (0/1) for whether the script has data sink destinations. Computed as <code>int(len(sink_data) > 0)</code> .
collection_method_diversity	Int	Number of different data sink methods used by the script. Computed as <code>len(sink_data)</code> .
SPECIFIC TRACKING CAPABILITIES		
tracks_mouse	Binary	Binary indicator (0/1) for whether the script tracks mouse events. Detected by checking for 'MouseEvent' in behavioral APIs.
tracks_keyboard	Binary	Binary indicator (0/1) for whether the script tracks keyboard events. Detected by checking for 'KeyboardEvent' in behavioral APIs.
tracks_touch	Binary	Binary indicator (0/1) for whether the script tracks touch events. Detected by checking for 'TouchEvent' or 'Touch.' in behavioral APIs.
tracks_pointer	Binary	Binary indicator (0/1) for whether the script tracks pointer events. Detected by checking for 'PointerEvent' in behavioral APIs.
COMPLEXITY CLASSIFICATION		
complexity_tier	Int	Categorical classification of script complexity: 0 (no APIs), 1 (≤ 5 APIs), 2 (6-15 APIs), 3 (> 15 APIs).
BALANCE METRICS		
is_behavioral_heavy	Binary	Binary indicator (0/1) for scripts focused on behavioral tracking. Computed as <code>int(total_behavioral > total_fp and total_behavioral > 5)</code> .
is_fp_heavy	Binary	Binary indicator (0/1) for scripts focused on fingerprinting. Computed as <code>int(total_fp > total_behavioral and total_fp > 5)</code> .
is_balanced_tracker	Binary	Binary indicator (0/1) for scripts with balanced tracking approaches. Computed as <code>int(abs(total_behavioral - total_fp) <= 3 and total_apis > 5)</code> .

We created 35 features (shown in Table 5.2) to identify behavioral biometric scripts. These features look at different aspects of how scripts collect user data:

- **Data Aggregation (10 features):** How scripts combine and process collected data. Scripts that aggregate lots of data are more likely to be doing behavioral biometrics.
- **Focus Areas (2 features):** Whether scripts focus more on behavioral tracking (like mouse movements) or fingerprinting (like device characteristics).
- **User Interaction Tracking (2 features):** How many different types of user actions the script monitors — mouse clicks, keyboard typing, touch gestures, etc.
- **Advanced Tracking (4 features):** Whether scripts track precise details like exact mouse coordinates, timing patterns, or device motion sensors.

- **Fingerprinting Methods (5 features):** What techniques scripts use to identify devices — canvas rendering, audio analysis, screen properties, etc.
- **Collection Intensity (2 features):** How aggressively scripts collect data and what types they prioritize.
- **Data Transmission (2 features):** Whether and how scripts send collected data to external servers.
- **Specific Capabilities (4 features):** Simple yes/no indicators for tracking mouse, keyboard, touch, and pointer events.
- **Overall Complexity (4 features):** How sophisticated the script is overall and whether it balances different tracking approaches.

We consider these vendor-agnostic features to help identify behavioral biometric scripts.

Feature Selection

To optimize model performance and maintain generalization capabilities, we employed a vendor-aware, three-step feature selection process:

- **Variance Thresholding:** Features exhibiting extremely low variance (< 0.01) were eliminated as they contribute minimally to differentiating between classes.
- **Statistical Significance Testing (ANOVA F-test):** Features were ranked based on their ability to statistically discriminate between behavioral biometric and non-behavioral scripts. The top-ranked features were retained for further evaluation.
- **Random Forest Feature Importance:** A Random Forest classifier was trained to evaluate each feature’s predictive power

Features with the highest predictive importance were selected for the final modeling stage. This vendor-aware approach split data by vendors (explained in Section 5.3.3), ensuring that features selected were genuinely predictive and not biased towards vendors seen during training. Importantly, all selection criteria were applied strictly to training data, preventing leakage and ensuring an unbiased evaluation.

Validation of Vendor Agnosticism

To further ensure generalizability, we rigorously evaluated correlations between each feature and individual vendors. Features showing a high correlation (≥ 0.3) with specific vendors were flagged as vendor-specific. We prioritized vendor-agnostic features with low correlation scores, preserving generalization and reducing the risk of model overfitting.

Final Features

Table 5.3: Top Twelve Important Features for Behavioral Biometric Script Detection (Balanced Dataset see Section 5.3.3)

Feature Name	Importance	Description
<code>total_aggregation_count*</code>	0.1999	Sum of behavioral and fingerprinting API aggregation counts. Computed as <code>behavioral_api_agg_count + fp_api_agg_count</code> . Indicates the total volume of API calls that involve data aggregation operations, which is a strong indicator of comprehensive data collection.
<code>max_api_aggregation_score*</code>	0.1910	Maximum aggregation score among all APIs accessed by the script. Set to 0 if the original value is NaN or -1. Represents the highest level of data aggregation complexity performed by any single API call, indicating sophisticated data processing capabilities.
<code>fp_approach_diversity</code>	0.1314	Sum of binary indicators for different fingerprinting approaches: <code>navigator</code> , <code>screen</code> , <code>canvas</code> , and <code>audio</code> APIs. Computed as <code>uses_navigator_fp + uses_screen_fp + uses_canvas_fp + uses_audio_fp</code> . Measures the diversity of fingerprinting techniques used by a script.
<code>collection_intensity</code>	0.0898	Total API access count divided by the number of unique APIs accessed. Computed as <code>total_accesses / max(total_apis, 1)</code> . Indicates how intensively each API is used, with higher values suggesting more aggressive and repeated data collection patterns.
<code>complexity_tier</code>	0.0854	Categorical classification of script complexity based on total API usage: 0 (no APIs), 1 (≤ 5 APIs), 2 (6-15 APIs), 3 (> 15 APIs). Represents overall tracking sophistication level and scope of data collection operations.
<code>tracks_coordinates</code>	0.0761	Binary indicator (0/1) for whether the script tracks mouse/pointer coordinates. Detected by checking for coordinate APIs like <code>'clientX'</code> , <code>'clientY'</code> , <code>'screenX'</code> , <code>'screenY'</code> , <code>'pageX'</code> , <code>'pageY'</code> in behavioral source APIs. Essential for precise behavioral pattern analysis and user interaction mapping.
<code>uses_canvas_fp</code>	0.0703	Binary indicator (0/1) for whether the script uses Canvas or WebGL APIs for fingerprinting. Detected by checking for <code>'Canvas'</code> or <code>'WebGL'</code> in fingerprinting source APIs. Canvas fingerprinting creates unique browser signatures by rendering graphics and measuring pixel-level differences.
<code>uses_screen_fp</code>	0.0337	Binary indicator (0/1) for whether the script uses Screen APIs for fingerprinting purposes. Detected by checking for <code>'Screen.'</code> in fingerprinting source APIs. Screen fingerprinting collects display characteristics like resolution, color depth, and pixel ratio for device identification.

Continued on next page

Table 5.3 – continued from previous page

Feature Name	Importance	Description
<code>behavioral_api_agg_count*</code>	0.0276	Count of behavioral APIs that perform aggregation operations. Set to 0 if the original value is NaN or -1. Specifically measures how many behavioral tracking APIs involve data aggregation, indicating sophisticated behavioral data processing.
<code>interaction_diversity</code>	0.0264	Count of different user interaction types monitored (mouse, keyboard, touch, pointer, device, wheel, focus). Computed by analyzing behavioral source APIs for event type patterns. Higher values indicate comprehensive user monitoring across multiple input modalities.
<code>tracks_device_motion</code>	0.0191	Binary indicator (0/1) for whether the script accesses device motion or orientation APIs. Detected by checking for 'DeviceMotion' or 'DeviceOrientation' in behavioral source APIs. Used for mobile device fingerprinting and advanced behavioral analysis on touch devices.
<code>sophistication_score</code>	0.0190	Sum of three binary indicators: <code>tracks_coordinates</code> , <code>tracks_timing</code> , and <code>tracks_device_motion</code> . Computed as <code>tracks_coordinates + tracks_timing + tracks_device_motion</code> . Measures the technical sophistication of behavioral tracking implementation with scores ranging from 0-3.
* Requires successful Program Dependency Graph construction		

Table 5.4: Top Twelve Features for Behavioral Biometric Script Detection (Imbalanced Dataset, see Section 5.3.3)

Feature Name	Importance	Description
<code>fp_approach_diversity*</code>	0.1981	Sum of binary indicators for different fingerprinting approaches: navigator, screen, canvas, and audio APIs. Computed as <code>uses_navigator_fp + uses_screen_fp + uses_canvas_fp + uses_audio_fp</code> . Measures the diversity of fingerprinting techniques used by a script.
<code>total_aggregation_count*</code>	0.1281	Sum of behavioral and fingerprinting API aggregation counts. Computed as <code>behavioral_api_agg_count + fp_api_agg_count</code> . Indicates the total volume of API calls that involve data aggregation operations.
<code>uses_canvas_fp</code>	0.1208	Binary indicator (0/1) for whether the script uses Canvas or WebGL APIs for fingerprinting. Detected by checking for 'Canvas' or 'WebGL' in fingerprinting source APIs. Canvas fingerprinting creates unique browser signatures.
<code>uses_audio_fp</code>	0.1075	Binary indicator (0/1) for whether the script accesses audio-related APIs for fingerprinting purposes. Detected by checking for 'Audio' in fingerprinting source APIs. Audio fingerprinting is a distinctive tracking technique.
<code>max_api_aggregation_score</code>	0.1001	Maximum aggregation score among all APIs accessed by the script. Set to 0 if the original value is NaN or -1. Represents the highest level of data aggregation complexity performed by any single API call.
<code>sophistication_score</code>	0.0939	Sum of three binary indicators: <code>tracks_coordinates</code> , <code>tracks_timing</code> , and <code>tracks_device_motion</code> . Computed as <code>tracks_coordinates + tracks_timing + tracks_device_motion</code> . Measures the technical sophistication of behavioral tracking implementation.
<code>collection_intensity</code>	0.0869	Total API access count divided by the number of unique APIs accessed. Computed as <code>total_accesses / max(total_apis, 1)</code> . Indicates how intensively each API is used, with higher values suggesting more aggressive data collection.
Continued on next page		

Table 5.4 – continued from previous page

Feature Name	Importance	Description
<code>behavioral_api_agg_count</code>	0.0574	Count of behavioral APIs that perform aggregation operations. Set to 0 if the original value is NaN or -1. Specifically measures how many behavioral tracking APIs involve data aggregation.
<code>has_both_aggregation_types*</code>	0.0312	Binary indicator (0/1) for whether the script performs both behavioral and fingerprinting API aggregation. Computed as <code>int(behavioral_agg > 0 and fp_agg > 0)</code> . Indicates comprehensive data aggregation capabilities.
<code>interaction_diversity</code>	0.0193	Count of different user interaction types monitored (mouse, keyboard, touch, pointer, device, wheel, focus). Computed by analyzing behavioral source APIs for event type patterns. Higher values indicate comprehensive user monitoring.
<code>tracks_timing</code>	0.0128	Binary indicator (0/1) for whether the script captures timing information. Detected by checking for <code>'timeStamp'</code> or <code>'interval'</code> in behavioral source APIs. Important for analyzing user behavior patterns over time.
<code>tracks_coordinates</code>	0.0124	Binary indicator (0/1) for whether the script tracks mouse/pointer coordinates. Detected by checking for coordinate APIs like <code>'clientX'</code> , <code>'clientY'</code> , <code>'screenX'</code> , <code>'screenY'</code> , <code>'pageX'</code> , <code>'pageY'</code> in behavioral source APIs. Essential for behavioral analysis.
* Requires successful Program Dependency Graph construction		

After applying our feature selection process, we reduced the initial set from 35 (shown in Table 5.2) to twelve features. Table 5.3 shows all twelve features we use for our final balanced classifier (see Section 5.4). Table 5.4 shows the features used to train our imbalanced classifiers in Section 5.3.4.

Validation confirmed these selected features maintained nearly identical predictive performance compared to the complete set (ROC AUC decreased marginally by 0.02%), demonstrating that the feature reduction was effective and justified.

5.3.3 Supervised Training Methodology

Given our set of selected features, we now aim to construct a classifier that can capture both linear and non-linear relationships within the data. As mentioned in Section 5.2, we have a dataset with 232 behavioral biometric scripts (positives) and 1,997 non-behavioral biometric scripts (negatives). Since our dataset consists of only 2,229 rows, we lean towards supervised models, as unsupervised models generally require more training data to be effective. Given the unbalanced nature of our dataset (more negatives than positives), we aim to experiment with two different classifiers: one trained on a balanced dataset, in contrast with one trained on an imbalanced dataset. The imbalanced dataset consists of 232 positives and 1,997 negatives. We construct our balanced dataset by randomly sampling 232 negatives from the total of 1,997, with the goal of having an equal number of positives and negatives.

Data Splitting

Before we can train our classifier, we have to split the data into a training and a test set. Our dataset of 232 positives consists of scripts from 18 unique vendors (scripts from the same vendor have very similar collection patterns). There are some vendors with multiple scripts (Iovation with 81 and Forter with 53 scripts), while for other vendors, we only have one script (ThreatMark, Callsign, etc.). This vendor imbalance within the positive set is due to a lack of datasets for behavioral biometric scripts. We suggest that the imbalance in our dataset is somewhat representative of the availability of each vendor’s scripts on the web, i.e., some vendors have more clients (e.g., Forter, BioCatch) than others.

Given such a dataset, if we split scripts based on a simple vendor-unaware 70-30 stratified split (where 70% of positive scripts go to the training set and 30% of positives go to the test set), we will have the problem of vendor data leakage across the training and test set, yielding optimistic model performance. Since our goals of classification are twofold: identifying scripts from both existing and new vendors, we cannot use a purely vendor-unaware data split methodology.

One alternative is to keep all scripts from a particular vendor in either the training or test set. This vendor-aware split would ideally counteract the existing data leakage across training and test sets. However, given our skewed distribution of data, where some vendors have more scripts than others, this would mean we must always put the dominant vendors (vendors who have the most scripts) in the training data. The issue with this approach is the training data will contain scripts from very few vendors, which may make it hard for the model to generalize and detect new vendors with ease. Additionally, with this method, we will only be able to evaluate how well the model can find scripts from new vendors, but we will not be able to learn how the model detects scripts from existing vendors as mentioned in our classification goals in Section 5.1.

We adopt a hybrid approach. First, we segment the vendors into three categories:

1. High-frequency vendors (≥ 20 scripts per vendor)
2. Medium-frequency vendors (5 – 20 scripts per vendor)
3. Low-frequency vendors (< 5 scripts per vendor)

Next, we place scripts from each vendor into the training and test set with different rules for each category. For high-frequency vendors, we place 70% of their scripts in the training set, while the remainder goes to the test set. Contrastingly, for medium-frequency and

low-frequency vendors, we place all scripts from the vendor with 60-40 and 50-50, splits respectively. All the scripts for a medium or low-frequency vendors can be in either the test or train set but not both. This means that for each medium vendors all its scripts will be placed in the training set with 60% probability and the test set with 40% probability (50-50 for low-frequency vendors). The final split consists of approximately 70% all positive (behavioral biometrics) scripts in the training set, while the remainder goes to the test set.

The advantage of this approach is we can test our classifier’s ability on both classification goals mentioned in Section 5.1 and offer scripts belonging to a larger number of vendors in the training set, while still maintaining scripts from unseen vendors in the test set, minimizing data leakage.

Handling Class Imbalance

In our dataset, we face two distinct forms of imbalance:

1. **Class Imbalance:** There are significantly fewer behavioral biometric scripts (positives) compared to non-behavioral biometric scripts (negatives).
2. **Vendor Imbalance:** Within the positive scripts, some vendors have a considerably larger number of scripts compared to others.

To effectively address these imbalances, we apply a two-tiered class weighting approach during model training.

Class-level Weighting: To counteract the general imbalance between positive and negative samples, we apply a standard **balanced weighting scheme**, where the penalty for misclassifying positive examples is greater, thus ensuring the model prioritizes correctly identifying the minority (behavioral biometric) class.

Vendor-level Weighting: To address imbalance across different behavioral biometric vendors, we introduce a customized inverse-frequency weighting strategy. Vendors with fewer scripts receive proportionally higher weights, preventing the model from overly biasing towards the more prevalent vendors.

Specifically, we compute weights for each vendor as inversely proportional to the square root of their frequency in the training set, ensuring a gentler weighting distribution that avoids overly aggressive adjustments:

$$w_v = \frac{1}{\sqrt{n_v}} \quad (5.1)$$

where:

- w_v is the initial weight for vendor v ,
- n_v is the number of training scripts from vendor v .

To normalize these weights so that the sum of all vendor weights equals the number of vendors (ensuring average vendor weight is one), we apply the:

$$w'_v = w_v \cdot \frac{|V_{pos}|}{\sum_{\text{vendor}_i \in V_{pos}} w_{\text{vendor}_i}} \quad (5.2)$$

where:

- w'_v is the normalized weight for vendor v ,
- w_v is the initial weight for vendor v (from Equation 1),
- V_{pos} is the set of unique vendors that have at least one positive sample in the training set,
- $|V_{pos}|$ is the number of unique vendors with positive samples,
- vendor_i represents each individual vendor in the set V_{pos} ,
- w_{vendor_i} is the initial weight for vendor i .

This normalization ensures that the average weight across all vendors with positive samples equals one, maintaining balanced influence while preventing any single vendor from dominating the training process.

Thus, each positive training sample x_i belonging to vendor v receives a sample weight $s_i = w'_v$, while negative samples remain at a default weight of 1.

In summary, our final sample weighting strategy can be defined as:

$$s_i = \begin{cases} w'_v, & \text{if } y_i = 1 \text{ and } \text{vendor}(x_i) = v \in V_{pos} \\ 1, & \text{if } y_i = 0 \end{cases} \quad (5.3)$$

This vendor-level weighting operates in combination with the classifier’s built-in class balancing (via `class_weight=‘balanced’`), which ensures positive samples receive higher overall importance than negative samples (this setting is only enabled when the dataset is imbalanced). The vendor weights (w'_v) specifically address imbalance within the positive class, where some vendors may receive weights less than 1 to prevent over-representation, while others receive weights greater than 1 to boost under-representation. This two-tier approach effectively balances both the broader class imbalance and the more nuanced vendor imbalance, resulting in improved model robustness and generalization across vendors.

Hyperparameter Tuning

Given the relatively small size of our dataset, we maximize data utilization by employing nested cross-validation (nested CV) for hyperparameter selection rather than using traditional train-validation-test splits, which would further reduce the already limited training data available for model development.

In supervised machine learning, selecting optimal hyperparameters is critical to maximizing model performance. However, traditional hyperparameter optimization can lead to overly optimistic performance estimates if hyperparameters are tuned and tested on the same data—a phenomenon known as hyperparameter optimization bias. To mitigate this bias and obtain reliable estimates of generalization performance, we employed nested cross-validation (nested CV).

When tuning hyperparameters, single-layer cross-validation (CV) can result in models being inadvertently tailored specifically to the evaluation set. This leads to overly optimistic performance estimates because the hyperparameters effectively “see” the evaluation data during the tuning process. Nested CV addresses this issue by separating hyperparameter tuning from performance evaluation into two distinct, nested loops:

- **Inner Loop:** Performs hyperparameter tuning through CV.
- **Outer Loop:** Provides an unbiased assessment by evaluating the hyperparameter-optimized models on data unseen during the tuning phase.

Formally, nested CV involves two nested loops:

Inner Loop (Hyperparameter Selection):

$$\theta_k^* = \arg \max_{\theta \in \Theta} \left[\frac{1}{K_{\text{inner}}} \sum_{j=1}^{K_{\text{inner}}} \text{AUC}(\text{Model}_{\theta}, \text{Val}_j^{\text{inner}}) \right] \quad (5.4)$$

Outer Loop (Performance Evaluation):

$$\text{Nested CV Score} = \frac{1}{K_{\text{outer}}} \sum_{k=1}^{K_{\text{outer}}} \text{AUC}(\text{Model}_{\theta_k^*}, \text{Test}_k^{\text{outer}}) \quad (5.5)$$

where:

- K_{outer} is the number of folds in the outer cross-validation loop,
- K_{inner} is the number of folds in the inner cross-validation loop,
- Θ is the hyperparameter search space,
- θ_k^* represents the optimal hyperparameters selected for outer fold k ,
- $\text{Test}_k^{\text{outer}}$ is the test set for outer fold k (unseen during hyperparameter tuning),
- $\text{Val}_j^{\text{inner}}$ is the validation set for inner fold j (used for hyperparameter selection),
- Model_{θ} represents a model trained with hyperparameter configuration θ ,
- $\text{AUC}(\cdot)$ denotes the Area Under the ROC Curve evaluation metric (Defined in Equation 5.6).

The inner loop selects optimal hyperparameters through cross-validation on the training portion of each outer fold, while the outer loop evaluates these optimized models on completely held-out test data, providing an unbiased estimate of generalization performance.

In practice, this means that for each of the K_{outer} outer folds, we first use the remaining data to perform a complete hyperparameter search via inner cross-validation (Equation 5.4), identifying the best hyperparameter configuration θ_k^* based on average AUC performance across K_{inner} inner folds. We then train a model using these optimal hyperparameters on all the training data for that outer fold and evaluate it on the corresponding held-out test set. The final nested CV score (Equation 5.5) averages these K_{outer} unbiased performance estimates, providing a robust measure of the model’s true generalization capability.

Stratified K-Fold Splitting for Class Imbalance: To preserve class distribution during splitting, we employed Stratified K-Fold Cross-Validation within the inner hyperparameter tuning loops. While our sample weighting strategy addresses class imbalance during model training, stratification ensures that each fold maintains roughly the same distribution of positive (behavioral biometric) and negative (non-behavioral biometric) scripts as the train

and test set during the cross-validation process itself. This approach prevents individual folds from having extremely skewed class distributions that could lead to unreliable hyperparameter selection, particularly in cases where a fold might contain very few or no positive samples.

In the inner CV loop for hyperparameter tuning, we deliberately used a vendor-unaware split. This means scripts from the same vendor could appear in both training and validation sets during hyperparameter optimization. Although this approach introduces a risk of vendor-specific information leakage and potentially optimistic AUC scores, the inner loop’s primary purpose is consistent hyperparameter selection across models. The final unbiased evaluation occurs in the vendor-isolated outer loop, where vendors are strictly separated. Additionally, attempting to do a vendor-aware split in the inner splits may result in very few scripts being in the training set for each fold.

To evaluate hyperparameter performance during tuning, we used **Area Under the ROC Curve (AUC-ROC)** as the scoring metric. AUC-ROC measures how well a model ranks true positive scripts higher than false positives, regardless of any specific classification threshold. This metric is particularly suitable for imbalanced datasets because it assesses overall ranking capability rather than accuracy at a single threshold:

$$\text{AUC-ROC} = P(\text{rank}(x_{\text{positive}}) > \text{rank}(x_{\text{negative}})) \quad (5.6)$$

In practical terms, AUC-ROC aligns well with our ultimate goal: reliably distinguishing behavioral biometric scripts from non-behavioral biometric scripts across different vendors and under varying threshold settings.

We systematically performed hyperparameter tuning using nested CV as follows:

1. **Model Selection:** Multiple classification algorithms were considered, including Random Forest, Gradient Boosting, LightGBM, Logistic Regression, and Support Vector Machines (SVM). Each algorithm had its hyperparameter grid.
2. **Inner CV for Tuning:** For each model, hyperparameters were optimized using a GridSearchCV with a StratifiedKFold inner loop. Specifically, hyperparameters were selected based on maximizing AUC-ROC:
3. **Outer CV for Unbiased Evaluation:** The optimal hyperparameters obtained from the inner loop were then evaluated in the outer loop, which provided an unbiased estimate of the model’s generalization performance. These parameters are mentioned in Appendix [B.2](#).

4. **Vendor-Aware Final Evaluation:** After nested CV, the final model performance was evaluated using a vendor-isolated test set, ensuring a realistic measure of generalization to entirely unseen vendors, simulating real-world conditions where new vendors with distinct behavioral patterns might emerge.

This strategy provides strong insights into the model’s true generalization capabilities.

5.3.4 Results

We trained and evaluated several classification models using two distinct datasets: a balanced dataset, containing equal numbers of behavioral biometric (positive) and non-behavioral biometric (negative) scripts, and an imbalanced dataset, which reflects the real-world scenario where negatives significantly outnumber positives (These datasets are the same ones we have previously mentioned in Section 5.3.3). We systematically compared each model’s performance on both datasets, considering metrics including accuracy, ROC-AUC, precision, recall, F1-score, and generalization to unseen vendors.

- **Precision:** The proportion of scripts predicted as behavioral biometrics that are actually behavioral biometrics. High precision means a low false positive rate—fewer non-behavioral biometric scripts incorrectly flagged as behavioral biometric.
- **Recall (Sensitivity):** The proportion of actual behavioral biometric scripts that are correctly identified by the model. High recall means low false negative rate—fewer behavioral biometric scripts missed by the detection system.
- **Accuracy:** The overall proportion of scripts (both behavioral biometric and non-behavioral biometric) correctly classified by the model.
- **F1-Score:** The harmonic mean of precision and recall, providing a single metric that balances both concerns. Particularly useful when dealing with class imbalance.
- **ROC-AUC:** Area Under the Receiver Operating Characteristic curve, measuring the model’s ability to distinguish between behavioral biometric and non-behavioral biometric scripts across all classification thresholds.

Evaluation on the Imbalanced Dataset

The following table summarizes the performance of models trained and evaluated on the imbalanced dataset, which better represents real-world conditions:

Table 5.5: Model Evaluation Results (Imbalanced Dataset)

Model	Accuracy	ROC AUC	Precision	Recall	F1-Score	Seen Recall	Unseen Recall
Gradient Boosting	0.9777	0.9823	0.97	0.92	0.94	0.9623	0.5455
LightGBM	0.9777	0.9758	0.96	0.92	0.94	1.0000	0.5000
Random Forest	0.9763	0.9681	0.97	0.91	0.94	0.9811	0.4545
XGBoost	0.9763	0.9785	0.97	0.91	0.94	0.9623	0.5000
SVM (RBF)	0.9703	0.9777	0.92	0.92	0.92	0.9811	0.5909
Logistic Regression	0.9496	0.9726	0.86	0.91	0.88	0.9623	0.5909
Naive Bayes	0.9362	0.9713	0.83	0.86	0.85	0.8302	0.5909

Seen recall refers to the recall of the model on positives from vendors whose scripts are part of the training data. For example, there are Iovation scripts in both the training and test set. The model may find it easier to classify Iovation scripts in the test set as it has seen similar scripts from the same vendor in its training data. Unseen recall refers to recall on totally unseen scripts from vendors that are not included in the training data. The unseen recall is especially volatile on our small datasets. If the model cannot detect a vendors pattern it will misclassify many scripts from the same vendor resulting in a drastic change in overall unseen vendor recall. We believe any value above 70% is good for our evaluations (the model generalizes well to most vendors but misses scripts from a few vendors).

Evaluation on the Balanced Dataset

The following table summarizes the model results on the balanced dataset:

Table 5.6: Model Evaluation Results (Balanced Dataset)

Model	Accuracy	ROC AUC	Precision	Recall	F1-Score	Seen Recall	Unseen Recall
Random Forest	0.932	0.977	0.95	0.92	0.93	1.000	0.727
Gradient Boosting	0.932	0.976	0.95	0.92	0.93	1.000	0.727
LightGBM	0.926	0.975	0.94	0.91	0.93	0.981	0.727
XGBoost	0.926	0.968	0.94	0.91	0.93	0.981	0.727
SVM (RBF)	0.892	0.963	0.93	0.85	0.89	0.962	0.591
Logistic Regression	0.892	0.956	0.92	0.87	0.89	0.962	0.636
Naive Bayes	0.885	0.945	0.90	0.87	0.88	0.981	0.591

Model Selection and Justification

Based on our comprehensive evaluation across both balanced and imbalanced datasets, we selected **Random Forest trained on the balanced dataset** as our final model for the following reasons:

1. **Vendor Generalization:** The balanced dataset Random Forest demonstrates good generalization to unseen vendors with a recall of 0.727, representing a 60% improvement over the imbalanced dataset’s 0.455. This improvement in detecting scripts from previously unseen vendors is important for a security application where new malicious actors constantly emerge. The model achieves this while maintaining perfect recall (1.000) on seen vendors, indicating it has learned robust, generalizable patterns rather than vendor-specific signatures.
2. **Theoretical Justification for Balanced Training:** While the concern about “not seeing enough negatives” in balanced training is valid, our results demonstrate that quality matters more than quantity. In our balanced dataset, the number of negative and positive examples is identical, but this means the ratio of negative to positive examples is smaller than what would be observed in the real world. Despite this reduced ratio, the imbalanced dataset suffers from a fundamental limitation in Random Forests: each tree’s bootstrap sample reflects the original imbalance, often resulting in trees that see very few or no positive examples. This leads to an ensemble biased toward memorizing majority class patterns. The balanced approach ensures every tree learns from a representative mix of both classes, leading to better generalization even with the artificially reduced negative-to-positive ratio.
3. **ROC-AUC Performance:** The balanced Random Forest achieves a ROC-AUC of 0.977, compared to 0.9681 for the imbalanced version. This improvement indicates better performance across all possible classification thresholds, suggesting the balanced model has learned a more robust decision boundary. The higher AUC despite lower overall accuracy (0.932 vs 0.976) reveals that the imbalanced model’s accuracy stems from correctly classifying easy negatives rather than learning meaningful distinctions.
4. **Production Deployment Considerations:** In our study, the ability to detect behavioral biometric scripts from new, previously unseen vendors is an important goal for comprehensive web measurement research. The 60% improvement in unseen vendor recall shows the improved performance of this model in identifying these data

collection practices. While the imbalanced dataset may better reflect web distribution, a model that fails to detect nearly 55% of scripts from new vendors poses an unacceptable gap in measurement coverage. We believe that the slightly lower overall accuracy on imbalanced test data is a worthwhile tradeoff for dramatically improved generalization in detecting behavioral biometric implementations across the web ecosystem.

5. **Comparison with Other Models:** While models like Gradient Boosting showed competitive performance, Random Forest’s combination of high unseen vendor recall, perfect seen vendor recall, and superior ROC-AUC on the balanced dataset makes it the optimal choice. Notably, models that performed well on balanced data (like Naive Bayes with 0.591 unseen recall on imbalanced data) showed unstable performance across dataset configurations, whereas Random Forest maintained robust performance characteristics (accuracy, AUC etc.).

The evidence overwhelmingly supports that the balanced dataset enables Random Forest to learn vendor-agnostic patterns of behavioral biometric scripts, making it the optimal choice for production deployment where detecting scripts from emerging vendors is critical.

5.4 Final Model

Following our comprehensive evaluation, we proceed to train a final production-ready model for deployment. Based on the comparative analysis demonstrating superior generalization capabilities, **Random Forest trained on a balanced dataset** was selected as our final model. To maximize the model’s capability to detect behavioral biometric scripts from both seen and unseen vendors, we construct a final balanced training set using all available positive examples and an equal number of randomly sampled negative examples. This approach ensures the model learns from the full diversity of behavioral biometric implementations across all 18 vendors while maintaining the balanced class distribution that our evaluation proved optimal for generalization. The hyperparameters employed for the final model are those optimized through nested cross-validation on balanced training subsets. This methodology is critical for maintaining model integrity, as the hyperparameter selection process used the same balanced approach that proved superior in our evaluation. By maintaining consistency between our evaluation methodology and final training approach, we ensure our production model will exhibit the same superior generalization capabilities demonstrated in our experiments. This balanced Random Forest model, with 72.7% recall on unseen vendors and perfect recall on seen vendors, will be deployed for our large-scale

study to detect behavioral biometric scripts on the web in Chapter 6. The model’s superior generalization capabilities ensure robust detection of both known and emerging behavioral biometric implementations.

5.5 Discussion

Our machine learning framework for identifying behavioral biometric scripts shows promising results, yet it is not without limitations. One key constraint is the limited size and diversity of our dataset. Despite careful curation, our dataset includes scripts from only 18 vendors, and even among these, the distribution is skewed — with a few vendors dominating the data. This introduces inherent bias in training and limits the ability of our models to generalize to entirely new vendors or novel data collection patterns.

Despite these constraints, our models perform well on two of our primary goals (as outlined in Section 5.1): (1) identifying scripts from known behavioral biometric vendors, and (2) discovering scripts from previously unseen vendors. Notably, our final Random Forest model achieved a test accuracy of 93.2% and an ROC-AUC of 97.7%, while maintaining strong precision of 95% to minimize false positives during large-scale deployment. The model demonstrated robust generalization with 72.7% accuracy on behavioral biometric scripts from completely unseen vendors, successfully detecting scripts from multiple distinct vendors including challenging cases with subtle behavioral patterns.

Manual analysis shows that misclassifications primarily occurred due to anomalous feature patterns. For instance, some vendor scripts exhibited significant deviation from typical behavioral biometric patterns, which we attribute to the variability in browser API triggers during dynamic analysis using VisibleV8. Since the behavioral data depends heavily on runtime execution and user-like interactions, missing triggered paths can lead to incomplete taint graphs, undermining the model’s prediction capability.

Interestingly, our model demonstrates excellent accuracy on scripts from known vendors that appear in both training and test sets. While this could raise concerns about overfitting, our manual inspection reveals otherwise. The intra-vendor collection patterns (in terms of API usage, aggregation intensity, and dataflow characteristics) tend to be consistent within vendors, especially among large vendors like Iovation or Forter. This allows the model to confidently detect these scripts. Given that a core classification goal was to recover scripts from known vendors with high accuracy, this behavior supports our use case rather than suggesting overfitting.

In designing our feature set, we aimed for vendor-agnostic signals to ensure generalizability. Features were mostly boolean or normalized values that reflect script behavior (e.g., frequency of access, flow to sink). However, due to vendor dominance in the data and the nature of API usage by specific vendors, some feature correlations to vendor identity remain. This is an acknowledged limitation. We emphasize that our current design reflects a best-effort feature selection given the dataset’s constraints and encourage future work to explore more robust feature engineering with larger, more balanced datasets.

Another approach for script classification would be to use string matching between known vendor scripts and newly discovered scripts. However, this method proves inadequate because companies often deploy multiple obfuscated versions of the same script or maintain different scripts with varying functionality. Such an approach would miss these variants, as we do not claim to have comprehensive coverage of all script versions from our known companies.

An important methodological consideration concerns our use of ROC-AUC as the primary evaluation metric during model selection. ROC-AUC can yield overly optimistic performance estimates on imbalanced datasets, since it incorporates true negatives in its calculation and may mask poor performance on the minority class. Area under the Precision-Recall curve would have provided a more appropriate evaluation framework for our imbalanced dataset, as it focuses specifically on positive class detection without being influenced by the abundance of true negatives. However, this limitation reinforces our findings rather than undermining them. Given that ROC-AUC tends to overestimate performance on imbalanced data, the substantial improvement we observed in our balanced model’s generalization to unseen vendors (72.7% versus 45.5% recall) suggests the true performance advantage is likely even more pronounced than our results indicate. Furthermore, since our final model selection favored the balanced dataset approach rather than the imbalanced one, any optimistic bias present in the imbalanced dataset evaluation did not influence our ultimate deployment decision.

Our final feature set (Table 5.3) requires taint analysis for three feature, making our classifier adaptable to graph construction failures and enabling detection of some heavily obfuscated scripts from vendors like CallSign using API usage patterns alone.

For future work, a promising direction is the exploration of contrastive learning with Siamese networks. If a significantly larger and more diverse dataset can be collected, deep learning models could learn a more robust representation of behavioral biometric scripts. Such architectures could differentiate subtle variations across scripts by learning similarity metrics, potentially enabling even better generalization across unknown vendors.

5.6 Conclusion

In this chapter, we presented a supervised machine learning approach to detect behavioral biometric scripts on the web. Our goal was to move beyond the limitations of manual identification and create a scalable, automated method capable of discovering both known and unknown vendors. To that end, we curated a labeled dataset of over 2,200 scripts, extracted a range of static features using taint-based analysis, and designed a vendor-agnostic feature set focused on behavioral patterns, fingerprinting techniques, and data aggregation.

We addressed key challenges in model training, including class and vendor imbalance, and validated our models using a vendor-aware hybrid split strategy to avoid data leakage. Our final model—a Random Forest classifier trained on the imbalanced dataset—offered the best trade-off between precision, ROC performance, generalization to unseen vendors, and real-world applicability. It achieved over 93% accuracy with an ROC-AUC of 97.7%, demonstrating good performance on the realistic class distribution expected in web-scale deployment.

While our dataset had limitations in size and vendor diversity, the results demonstrate that it is possible to reliably detect behavioral biometric scripts at scale using carefully engineered static features and ensemble classifiers. Our model’s strong performance on difficult cases and consistent behavior across different data distributions further support this claim. At the same time, failure cases—particularly those involving missing graph constructions—highlight the importance of dynamic coverage and runtime variability in feature extraction.

Overall, this work provides a foundation for large-scale detection of behavioral biometric scripts and opens up several promising directions for future work, including more robust feature extraction and the use of contrastive or deep learning models trained on larger, more diverse datasets. We now use this classifier to study the prevalence of behavioral biometric tracking across the web in the following chapter.

Chapter 6

Evaluation

6.1 Introduction

Behavioral biometrics rely on subtle user interactions, such as mouse movements, keyboard typing rhythms, and touch gestures, to uniquely identify users. These signals enhance traditional authentication methods and effectively distinguish legitimate users from automated bots, as previously discussed in Chapter 2. During our preliminary evaluation, we observed that behavioral biometric scripts were predominantly deployed by financial institutions, especially banks based in North America and Europe.

This observation raised an important question: **How widespread is the adoption of behavioral biometric services across the web today?** To answer this, we conducted two complementary large-scale measurement studies:

- **Bank-focused study:** We crawled approximately 9,500 U.S. banking websites, chosen from publicly available regulatory lists and industry rankings, to assess the prevalence of behavioral biometric implementations in financial institutions.
- **General-web study:** We analyzed the top 100,000 websites from the Chrome User Experience (CrUX) dataset [41]. This method follows the approach of previous large-scale web fingerprinting studies that have similarly evaluated tracking and fingerprinting practices across thousands of popular websites [46, 81, 1].

Each study was performed under two distinct scenarios:

1. **Landing-Page Crawl:** Analyzing only the homepage of each website to determine immediate exposure to behavioral biometric scripts.
2. **Login-Page Crawl:** Employing our specialized Login Crawler (LoginGPT, detailed in Chapter 4) to access login or authentication pages, where these scripts are more likely to be deployed.

By combining results from both crawling methods, we offer comprehensive insights into the prevalence and distribution of behavioral biometric services. Specifically, our bank-focused crawl provides a clear view of adoption rates within the critical sector of financial services in the U.S. Meanwhile, the general web crawl illustrates how often everyday users might encounter these services online.

In summary, among the U.S. banks for which our crawler found a login page, **15.8%** use behavioral biometrics, and among the 100K websites that our crawler found a login page for **1.79%** currently employ behavioral biometric scripts. The remainder of this chapter details our data collection strategies, crawling methodologies, script detection mechanisms, and a thorough analysis of our results.

6.2 Methodology

In this section, we will discuss the methodology for our large-scale study.

6.2.1 Dataset Construction & Justification

Our large-scale study is done in two parts. One part is using the top 100K domains from the CrUX list, and the other is using banks in the United States. In this section, we justify our selection of the CrUX dataset, while simultaneously describing the construction of our bank website dataset.

CrUX List

We use for our measurements the *CrUX Top 100 K* snapshot released in April 2025.¹ CrUX offers several advantages over legacy popularity lists such as Alexa, Tranco, or Cisco Umbrella. First, it is derived from real-user-measurement telemetry collected from consenting

¹Available via Google BigQuery at `bigquery-public-data.chrome_ux_report`. Documentation and access instructions at <https://developer.chrome.com/docs/crux/>

Chrome users, which makes it far less susceptible to bot inflation or rank manipulation than crawler-based lists [138]. Second, Google publishes fresh snapshots every month, so researchers can pin experiments to a precise point in time and still reproduce results years later; this longitudinal stability has already been leveraged in recent large-scale web studies [143, 142]. Third, CrUX enumerates *web origins* rather than arbitrary DNS names, reducing the incidence of parked domains and non-HTTP services that hamper automated crawls [137]. Finally, multiple top-tier security papers, including cookie-notice compliance analyses [18], crawler-effectiveness evaluations [148], and browser-fingerprinting work [143], have already adopted CrUX, signaling clear community acceptance and minimizing concerns about list bias. For these reasons, the April 2025 CrUX list offers the best balance of coverage, realism, and reproducibility for our thesis experiments.

Bank List

While the CrUX list is readily available via Google Chrome, there is no comprehensive list of banks for the U.S. Existing work [108, 37] uses an open-source list of bank websites, known as the Community Banker List [155]. While this list contains websites for many banks in the U.S. (categorized by states), we found it to have many broken links. Hence, we aimed to complement this list by using an official U.S. government website [53]. This is an up-to-date list of banks insured by the Federal Deposit Insurance Corporation [54], which displays banks for each state. We developed a custom crawler in Puppeteer [64] to scrape the list of all FDIC-insured banks. Next, we added this to the list of banks from the Community Banker website and removed any duplicates. The result was a comprehensive list of 9,502 banks in the U.S. [153].

6.2.2 Crawling & Classification Pipeline

Figure 6.1 shows a high-level representation of our pipeline. We perform two kinds of crawls: login-page crawl and a landing-page crawl. We will first describe the flow for the login-page crawl, which includes our specialized LLM-based crawler from Chapter 4.

We feed our list of domains to the Login Crawler backend API. The backend forwards each URL to a crawling worker that attempts to find login pages for one URL at a time (refer to Section 4.3.2 for a more detailed description of login page detection strategies). As soon as LoginGPT finds all login pages for a particular input domain, it forwards these domains to the next subsystem, VisibleV8 crawler. If, however, our crawler does not find any login pages for a given domain, that domain is skipped.

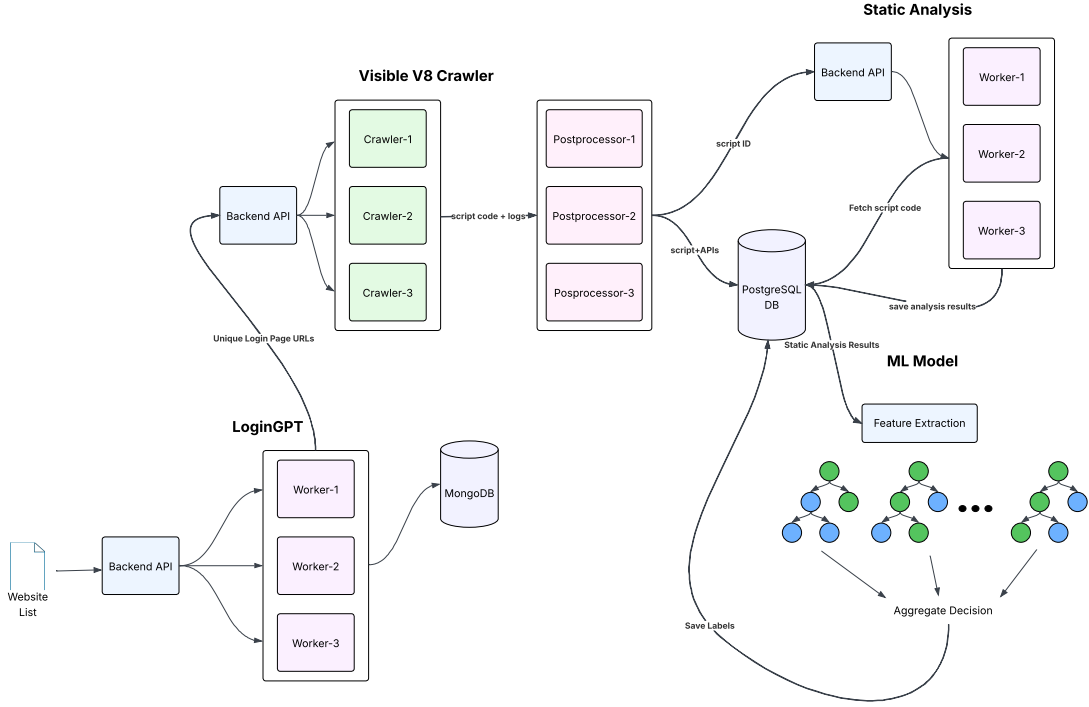


Figure 6.1: Evaluation pipeline system diagram

This crawler (described in Chapter 3), uses an instrumented version of Chrome (VisibleV8). It allows us to detect what browser APIs are triggered by each Javascript file on a web page. When a web page is visited, the browser stores all browser APIs that were accessed (e.g. MouseEvent.x, Navigator.userAgent) into raw log files. These raw log files are forwarded to the postprocessing workers for the VisibleV8 Crawler. These workers store all triggered browser APIs into a PostgreSQL database along with the corresponding script that invoked them. After the successful storage of all scripts for a web page and their corresponding APIs into the database, the postprocessor sends the ID of the corresponding scripts to the next subsystem (Static Analysis).

The Static Analysis subsystem receives a list of script IDs in its backend API, which forwards these IDs to static analysis workers. Each worker then fetches the script code associated with one script ID at a time. Next, we run our static analysis logic described in Chapter 3, on each script to extract features related to data flow and the number of relevant source and sink APIs collected. The worker then stores this information back into the PostgreSQL database.

Our machine learning model takes the static analysis results for each script and performs feature extraction. These features are the same ones used to train the model (described in Chapter 5). Next, it feeds these features to our Random Forest, which makes an aggregate decision using multiple decision trees and this decision is then stored back into our PostgreSQL database. The decision is whether a given script is doing behavioral biometrics or not.

The landing-page crawl is similar to the login-page crawl shown in Figure 6.1, with the exception that the domains are fed directly to the VisibleV8 Crawler instead of the Login Crawler.

We design our system specifically for scalability, ensuring each subsystem can independently scale according to available computational resources. Among these subsystems, the **Login Crawler** is notably time-intensive, often taking several minutes per domain to discover login pages. The **Static Analysis** component is similarly resource-demanding, which is configured to spend at most seven minutes per script. Given that a single domain may have multiple login pages, each containing several scripts, static analysis can quickly become a bottleneck. A naive solution—processing each subsystem sequentially, by first identifying login pages (LoginGPT), then extracting scripts and APIs (VisibleV8 Crawler), followed by static analysis, and finally applying machine learning—would be impractical for processing hundreds of thousands of scripts. To mitigate this, we adopt a streaming architecture: as soon as **LoginGPT** identifies login pages, they are immediately forwarded to the **VisibleV8 Crawler**. Likewise, once scripts from a login page are processed, they are promptly sent to the static analysis subsystem. This pipelined approach significantly accelerates large-scale analyses, and it forms the basis of the evaluation setup presented in this thesis.

6.3 Results

In this section we discuss the results of our two studies, one focused on banks in the U.S. and the more general web study using the CrUX list.

6.3.1 System Configuration

To ensure resource isolation and maximize throughput, we run each major subsystem—Login Crawler, VisibleV8 Crawler, and Static Analysis—on a separate physical machine. The

machine learning model is the only component not isolated in this way, as its resource requirements are minimal. For all our studies, the Login Crawler is configured with 40 parallel workers (compared to 25 in Section 4.4.1, but with identical hardware). The VisibleV8 Crawler and Static Analysis systems run on two identical high-performance servers, each equipped with 160 logical CPUs and 1 TB of RAM, using Intel Xeon E7-8870 processors. The VisibleV8 Crawler processes up to 60 websites in parallel, while the Static Analysis (which is more CPU-intensive) does 100 analyses in parallel. This setup allows the system to process login workflows and script analyses at scale without becoming bottlenecked by computing constraints.

In addition to this, we add telemetry and metric observation across our distributed setup. LoginGPT has detailed instrumentation for tracking LLM inference times as well as detailed traces using Prometheus [131] and Jaeger [157], visualized in Grafana [68]. We aggregate all logs from all our subsystems using Loki [69], making the tool ideal for production settings with good observability.

6.3.2 U.S. Banking Sector

We ran our pipeline on 9,502 banks from the list described in 6.2.1. This study was run in two settings: login-page based crawl and landing-page based crawl. We report the results below.

Table 6.1: Website Reachability Analysis (N = 9,502 bank websites)

Category	Count	Percentage
Overall Reachability		
Total Websites Studied	9,502	100%
Reachable Websites	7,177	75.5%
Unreachable Websites	2,325	24.5%
Error Breakdown		
DNS Lookup Failures	1,351	14.2%
403 Forbidden Errors	218	2.3%
404 Not Found Errors	190	2.0%
Crawler Timeouts	6	0.1%
Other Errors (400, 503, 409, etc.)	560	5.9%

Landing-Page Crawl

For the 9,502 bank domains, our crawlers found 29,450 scripts that collect behavioral data. **1,407** scripts were classified as behavioral biometrics by our Random Forest model, **727** of these we manually confirmed as doing behavioral biometrics from known providers. Of the remaining 700 scripts we ruled out 601 as not doing behavioral biometrics (these scripts collected little to no behavioral information), 79 scripts were heavily obfuscated so we could not make any assumptions nor could we tie them to any specific company. We tried to monitor the data collection and network requests for each script but that too gave no specific indicators, hence we labeled these scripts as *Unsure*. From the 727 scripts that are from known behavioral biometric providers, the majority belong to Iovation [79] (as shown in Table 6.2). In this dataset we found only one domain with two different behavioral biometric providers, bnh.com, hence a total of **435** unique domains use behavioral biometrics.

Table 6.1 shows a detailed breakdown of how many websites from our set of 9502 were reachable by our crawler. 75.5% of pages were accessible by our crawler, the remainder failed for a variety of reasons, including 400 and 500 series errors. However, 58.1% of failures were caused by DNS lookup failures, where the domain did not exist. We manually verified this by randomly sampling these domains and visiting them.

Overall we find for our Landing-Page Crawl that **6.1%** of the 7,177 banking domains that we were able to access use behavioral biometrics.

Table 6.2: Behavioral Biometric Scripts Distribution for 9,502 Banks via Landing-Pages

Company	Number of Domains	Example Domains
Iovation	413	firsticbank.com, usmetrobank.com
Cheq	26	mtb.com, unionsavings.com
BioCatch	15	wellsfargo.com, bankofamerica.com
Nudata	12	firstwesternbank.com
BehavioSec	4	citibank.com, investarbank.com
Forter	1	bnh.com
PerimeterX	1	bnh.com

Login-Page Crawl

The Login-Page Crawl attempts to find login pages for each domain; if our LoginGPT Crawler is unable to find a login page for a domain, that domain is skipped. Table 6.3

Table 6.3: Login Page and Behavioral Script Analysis

Category	Count	Percentage
Login Page Discovery		
Websites with Login Pages	5,315	74.1%
Total Login Pages Found	13,380	-
Average Login Pages per Website	2.5	-
Behavioral Script Detection		
Total Scripts Analyzed	52,900	-
Scripts Flagged as Behavioral Biometrics	3,243	(6.1% of total)
Scripts Confirmed Behavioral Biometrics	1,657	(51.1% of flagged)
Scripts Labeled as Unsure	1,295	(40.0% of flagged)
False Positives	291	(8.9% of flagged)
Domains with Behavioral Biometrics		
Domains with Login Pages	5,315	-
Domains that use Behavioral Biometrics	801	(15.1% of crawled domains)

shows a detailed breakdown of our login page discovery. We are able to reach 75.5% (7,177) of pages from our set of 9,502 as shown in Table 6.1), of these, we find **at least one** login page for 5,315 domains (**74.1%** of reachable domains). From these login pages (13,380 unique pages in total), we find 52,900 scripts that collect behavioral data. Our machine learning model flags 3,243 (6.1%) scripts as behavioral biometrics.

We manually verified these labels and found that 1,657 (51.1%) of these scripts are indeed from known behavioral biometric providers. 1,295 scripts were heavily obfuscated hence we could not link them to a specific company, hence we labeled them as *Unsure*. The remaining 291 scripts did not collect enough behavioral features so we labeled them as false positives.

After a detailed analysis we found scripts from 7 behavioral biometric vendors on these login pages (shown in Table 6.4). Some webpages like citi.com and wells Fargo.com had two behavioral biometric scripts from providers like BehavioSec, NuData, and BioCatch. Overall, we found that **841** unique domains from our set were using behavioral biometric services. This means that **15.8%** of the 5315 domains that we found login pages for, appear to be using behavioral biometric services.

In contrast to our Landing-Page Crawl, which visits pages from 7,177 domains in total, and finds behavioral biometric scripts on **6.1%** of these domains- our Login-Page crawl visits pages from 5,315 domains and finds behavioral biometric scripts on **15.8%** of these

Table 6.4: Behavioral Biometric Scripts Distribution for 9,502 Banks via Login-Pages

Company	Number of Domains	Example Domains
Iovation	595	creditonebank.com, americanstatebank.com
BehavioSec	113	truist.com, discover.com
NuData	88	busey.com, bankofpontiac.com
BioCatch	59	td.com, mtb.com
Cheq	16	unionsavings.com, belmontsavings.com
PerimeterX	2	bnh.com
PingOne	1	navebank.com

domains. Despite visiting pages from fewer domains, in our Login-Page Crawl, we find a much higher percentage of domains that use behavioral biometrics. This confirms our initial assumption in the case of banks, that behavioral biometric scripts are more prevalent on login pages versus landing pages.

6.3.3 CrUX Top 100K Domains

Next we ran our system on the top 100K domains to measure the prevalence of behavioral biometric scripts on the web, and how likely an average user is to encounter this service.

Table 6.5 shows a detailed breakdown of the domains that were reachable and prominent error cases for our study. Our crawlers successfully reached **85.1%** of the top 100K websites. We separate our failures into two categories: primary and secondary failures. Primary failures represent the most common and significant error types that prevented access to domains, accounting for the majority of failures (89.9% of all failures). These include fundamental issues like bot detection (403 errors), complete unresponsiveness (timeouts), domain resolution problems (DNS failures), and missing pages (404 errors). Secondary failures encompass less frequent but diverse technical issues that collectively account for the remaining 10.1% of failures, including various HTTP response problems, connection issues, server errors, and network-related problems.

The most prominent failure was due to 403 errors (permission denied). We suspect our crawler was unable to access the domain due to bot detection. The second most prominent error case was timeouts. Upon manual investigation, these are mostly due to the domain not loading. We verified this by running a random subsample of these websites on our local machines to see if loading these pages takes longer than 60 seconds. Many domains indeed did not load even past the 60-second mark. Other errors include DNS failures,

Table 6.5: Large-Scale Website Reachability Analysis (N = 100,000 domains)

Category	Count	Percentage
Overall Reachability		
Total Domains Analyzed	100,000	100%
Successfully Accessed	85,109	85.1%
Failed to Access	14,891	14.9%
Primary Failure Categories		
403 Forbidden	7,812	(52.5% of failures)
Timeout	2,989	(20.1% of failures)
DNS Resolution Failed	1,659	(11.1% of failures)
404 Not Found	917	(6.2% of failures)
Secondary Error Categories		
HTTP Response Failures	406	(2.7% of failures)
Connection Issues	269	(1.8% of failures)
5xx Server Errors	241	(1.6% of failures)
4xx Client Errors	188	(1.3% of failures)
SSL/TLS Errors	17	(0.1% of failures)
Other Network Errors	194	(1.3% of failures)

404, and 500 series errors. Despite CrUX listing actual web origins, DNS failures can still occur because domains may expire, go offline, or change DNS settings between the April 2025 snapshot and our crawling period. Additionally, temporary DNS issues and resolver differences between our crawler and typical Chrome users can cause resolution failures

Landing-Page Crawl

From the 85,109 domains successfully accessed during our CrUX Top 100K domain study, we analyzed a total of **162,876** scripts that collected behavioral data. Our machine learning model labeled **3,660** scripts as behavioral biometrics, out of which **519** scripts were manually verified as genuine behavioral biometric scripts belonging to known providers. Additionally, we were uncertain about **517** scripts due to heavy obfuscation or unclear collection patterns, thus labeling these scripts as *Unsure*. The remaining **2,624** scripts were determined to be false positives, as they collected minimal or no behavioral data.

The confirmed behavioral biometric scripts span **431** unique domains. Notably, some domains hosted scripts from more than one behavioral biometric provider. For example,

certain websites utilized scripts from PerimeterX alongside another behavioral biometric vendor. We hypothesize that in such instances, PerimeterX may primarily be deployed for bot detection, while the additional vendor might focus on credential stuffing and account takeover prevention. However, due to the opaque nature of script operations, we cannot definitively confirm this hypothesis.

Table 6.6 summarizes the distribution of confirmed behavioral biometric scripts across identified providers. PerimeterX leads with **165** domains, followed by Forter with **126** domains. Cheq is also prominent, appearing on **99** domains. Other notable providers include Iovation (**43** domains), BioCatch (**21** domains), and BehavioSec (**17** domains). We observe less prevalence for providers such as DataDome and PingOne (**9** domains each), GroupIB (**5** domains), and several providers including Accertify, Arkose Labs, Nudata, Sardine, Transmit Security, and F5 Risk Services (**3** domains each). The smallest presence is observed for Darwinium and Yofi (**2** domains each), and Feedzai and MCP Shield (**1** domain each).

In this analysis, our machine learning model also flagged a script from a behavioral biometric provider we were previously unaware of, known as MCP Shield, a UK-based service that provides real-time fraud detection by analyzing behavioral signals across mobile and web interactions using AI and machine learning [110].

Overall, our landing-page crawl of the top 100K websites reveals that **0.506%** (431 out of 85,109 reachable domains) utilize behavioral biometric scripts. This finding highlights that while behavioral biometrics are notably adopted by websites handling sensitive workflows, such as financial transactions or user authentication, they are not yet extensively widespread across the general web.

Login-Page Crawl

For our Login-Page Crawl, our Login Crawler found a total of **113,931** login pages across **53,308** unique domains. We restricted our crawl to only domains where our crawler successfully located at least one login page, skipping all others. On these login pages, we found **445,536** scripts that collect behavioral features, of which our model labeled **11,167** as behavioral biometric scripts. Upon manual verification, we confirmed that **1,686** of these scripts indeed perform behavioral biometrics. We marked **7,623** scripts as unsure due to heavy obfuscation or unclear collection patterns, while the remaining scripts were identified as false positives. Many of these false positives belonged to companies performing session replay or user-experience analytics, such as FullStory [60], Medallia [111], ByteDance [20],

Table 6.6: Behavioral Biometric Scripts Distribution for CrUX Top 100K via Landing-Pages

Company	Number of Domains	Example Domains
PerimeterX	165	fiver.com, wallmart.com, grubhub.com
Forter	126	docusign.com, popeyes.com, wish.com
Cheq	99	upwork.com, feetfinder.com
Iovation	43	clearscore.com, flirt4free.com
BioCatch	21	bankofamerica.com, citi.com
BehavioSec	17	fidelity.com, ebay.com
DataDome	9	vinted.com
PingOne	9	victoriasecret.com, potterybarnkids.com
GroupIB	5	betboom.ru, betsafe.pe
Accertify	3	jcpenny.com, grubhub.com
Arkose Labs	3	uber.com
Nudata	3	www.ipko.pl
Sardine	3	giftcards.com, getjobber.com
Transmit Security	3	secure.ally.com
F5 Risk Services	3	mybca.bca.co.id
Darwinium	2	apollo.io, getpostman.com
Yofi	2	dtlr.com, rhodeskin.com
Feedzai	1	tatrabank.sk
MCP Shield	1	360vuz.com

DynaTrace [42], Yandex [172], and Sentry [145], which collect some behavioral data but not for biometric identification or fraud prevention.

In total, we identified behavioral biometric scripts on **956** of the **53,308** domains visited, which amounts to approximately **1.79%**. This is a notable increase compared to our Landing-Page Crawl, where we found behavioral biometric scripts on **0.506%** (431 of 85,109) of domains. These findings confirm our initial assumption that behavioral biometric scripts are significantly more prevalent on login pages compared to landing pages.

Table 6.9 summarizes the distribution of behavioral biometric scripts from our Login-Page Crawl. Forter leads with **424** domains, followed by Cheq (**286** domains) and Iovation (**267** domains). BehavioSec (**198** domains), PerimeterX (**134** domains), and BioCatch (**113** domains) also have notable representation.

Additionally, we discovered **four more** previously unidentified behavioral biometric providers

Table 6.7: Behavioral Biometric (BB) Adoption Across Studies and Page Types

Crawl Type	Domains Visited	BB Domains (%)
CrUX Top 100K Websites		
Landing Page Crawl	85,109	431 (0.51%)
Login Page Crawl	53,308	956 (1.79%)
U.S. Banking Sector		
Landing Page Crawl	7,177	435 (6.1%)
Login Page Crawl	5,315	841 (15.8%)

in this crawl: GeeTest [61], JuicyScore [88], VeriSoul.ai [163], and F6 [49]. According to their company website, GeeTest combines CAPTCHA technology with behavioral biometrics, analyzing mouse movements, typing patterns, and timing data to distinguish humans from bots. JuicyScore’s website describes their fraud prevention approach through device fingerprinting and behavioral analytics, monitoring user interactions and behavioral patterns to assess authenticity. VeriSoul.ai states on their website that they detect fake accounts and fraud by analyzing device behavior, email intelligence, and user interaction patterns including mouse and touch behavior. F6’s website indicates they focus on touch pressure and interaction patterns, though our analysis also found it collecting mouse and keystroke data for user authentication.

6.3.4 Landing Versus Login Study Comparison

Table 6.7 summarizes our findings for both studies across landing and login pages. Several key insights emerge from this comparison.

For the U.S. banking sector, we observed a significantly higher adoption of behavioral biometric services on login pages (15.8%) compared to landing pages (6.1%). This difference aligns with our initial expectations and observations—banks primarily deploy behavioral biometric scripts on sensitive workflows such as user authentication to detect bots, fraudulent activities, and account takeovers. The landing pages typically serve more general informational purposes and are less frequently targeted by these security measures.

In the general CrUX Top 100K websites study, the overall adoption rates were substantially lower compared to the banking sector. Only 0.51% of landing pages and 1.79% of login pages contained behavioral biometric scripts. However, similar to the banking sector study, we found that login pages had a notably higher adoption rate, more than three times greater than the rate observed on landing pages. This supports our hypothesis that behavioral

biometric technologies are specifically favored in scenarios where sensitive user information or accounts are at stake.

Comparing the two studies, it is evident that behavioral biometrics are far more prevalent within specialized industries such as banking, where user account security is paramount. The general web adoption rates, while significantly lower, still demonstrate a clear trend toward increased usage on login workflows rather than landing pages. These findings reinforce our initial assumption that behavioral biometric scripts are primarily leveraged for protecting sensitive interactions and user authentication processes.

6.3.5 Comparison with Dynamic Taint Analysis Study

Table 6.8: Behavioral Biometric Script Prevalence Comparison

Page Type	Study	Domains Analyzed	BB Prevalence
Banking Login	Bara	462	4.55%
	Our Study	5,315	15.8%
Banking Landing	Bara	462	0.65%
	Our Study	7,177	6.1%
General Login	Bara	500	1.00%
	Our Study	53,308	1.79%
General Landing	Bara (Top 20K)	13,278	0.31%
	Our Study (Top 100K)	85,109	0.51%

BB = Behavioral Biometric

Both studies used LoginGPT for login page discovery

Our study achieved higher domain accessibility (85.1% vs 66.4%)

To provide context for our findings, we compare our results with the concurrent dynamic taint analysis study by Bara [11], which employed a complementary approach using modified FoxHound browser and dynamic crawling techniques. While our studies used different detection methodologies—static taint analysis with machine learning classification versus dynamic taint analysis—both investigations share common objectives and datasets that enable meaningful comparisons. Table 6.8 shows a detailed differential of our study compared with Bara’s.

Banking Sector Analysis

Both studies examined U.S. banking websites (from the same list of 9,502 banks that we produced), though at different scales. Bara analyzed 500 banking websites, finding behavioral biometric scripts on 21 login pages (4.55% of banking domains with discoverable login pages). Our larger-scale study of 5,315 banking domains with discoverable login pages revealed behavioral biometric adoption on 15.8% of these domains. While the absolute percentages differ, both studies consistently demonstrate that banking institutions represent a primary deployment sector for behavioral biometric technologies, with significantly higher adoption rates compared to general web domains.

General Web Prevalence

For general web analysis, Bara examined 500 websites from the Tranco top 20K list, discovering behavioral biometric scripts on 1.00% of domains with discoverable login pages and 0.65% of all analyzed domains. Our broader CrUX Top 100K study found behavioral biometric scripts on 1.79% of domains with discoverable login pages and 0.51% of all successfully accessed domains for landing pages. The prevalence rates align remarkably well despite the scale difference (500 vs. 53,308 domains with login pages), suggesting consistent deployment patterns across the web.

Additionally, comparing landing page prevalence across different scales, Bara’s analysis of the top 20K websites found behavioral biometric scripts on 0.31% of successfully crawled domains, while our CrUX Top 100K analysis identified adoption on 0.51% of accessible domains. This slight increase in prevalence when examining a broader set of popular websites may reflect the strategic deployment of these technologies on higher-traffic platforms.

Login Page Detection and Access Rates

Both studies utilized LoginGPT for login page discovery, ensuring consistency in page identification methodology. However, our studies achieved different domain accessibility rates: Bara successfully crawled 66.4% of their target websites (13,278 out of 20,000), while our CrUX-based study achieved 85.1% accessibility (85,109 out of 100,000 domains). This improvement likely reflects the inherent quality of the CrUX dataset, which enumerates actual web origins with real user traffic rather than potentially outdated or parked domains that may appear in traditional popularity rankings.

Detection Methodology Effectiveness

While both approaches successfully identified behavioral biometric scripts, they encountered different technical challenges. Bara reported difficulty tracking certain vendors like Forter due to dynamic taint analysis limitations, particularly with decision-tree-based obfuscated scripts where taint propagation was lost. In contrast, our static analysis approach successfully identified Forter scripts on 550 domains across both studies (126 domains in the CrUX landing page crawl and 424 domains in the login page crawl), demonstrating complementary strengths between the methodologies.

Our static analysis framework proved adaptable even when Program Dependency Graph construction failed for heavily obfuscated or extremely large scripts. In such cases, we leveraged alternative features including dynamically triggered API access patterns and collection intensity metrics captured during the VisibleV8 crawling phase. This multi-faceted feature extraction approach enabled our machine learning classifier to maintain high accuracy (93.2%) even when traditional taint flow analysis was insufficient.

Consistent Findings Across Methodologies

Despite methodological differences, both studies reach the same fundamental conclusion: behavioral biometric scripts are significantly more prevalent on login pages compared to landing pages. This consistency across different detection approaches strengthens confidence in the finding that organizations often deploy these technologies to protect sensitive authentication workflows rather than general browsing experiences.

Both studies also encountered similar operational challenges, including bot detection mechanisms that resulted in 403 Forbidden errors, timeout issues with unresponsive domains, and the strategic use of obfuscation by behavioral biometric providers to protect their intellectual property.

Vendor Coverage and Discovery

Our static analysis approach identified behavioral biometric scripts from many vendors across both studies, including five previously unknown providers (GeeTest, JuicyScore, VeriSoul.ai, F6, and MCP Shield). This vendor discovery capability demonstrates the effectiveness of our machine learning classification approach in generalizing beyond known vendor patterns, complementing the detailed flow analysis provided by dynamic taint tracking.

The convergence of findings between these independent methodologies—static taint analysis and dynamic taint analysis—provides robust evidence for the deployment patterns and prevalence rates of behavioral biometric technologies on the modern web.

6.4 Discussion

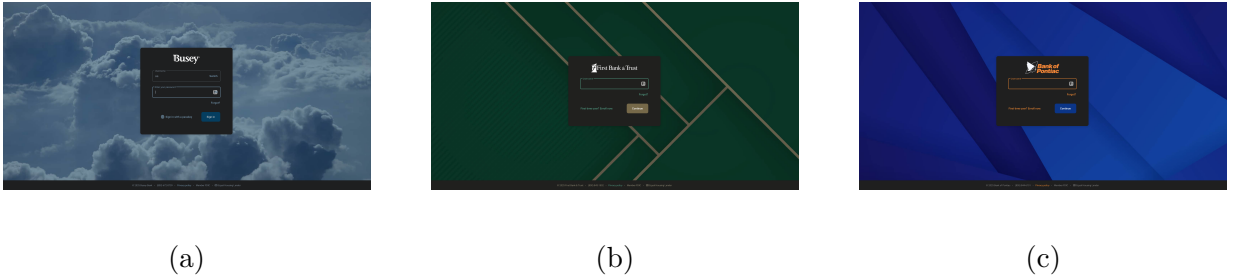


Figure 6.2: NuData authentication provider user interface across different webpages

Our large-scale study demonstrates the robustness and strong generalization capability of our machine learning model. Initially trained on scripts from 18 (we do not train on all 26 since we did not have websites where scripts from the other 5 providers were active) web behavioral biometric companies, the model successfully identified scripts from five previously unknown vendors, representing a notable increase of approximately 19% in the total known vendors (5 new from the original 26 for which we had found scripts in Table 2.1) and underscoring the efficacy of our vendor-agnostic feature engineering and selection methodology.

Despite careful preparations, we encountered access limitations during our crawls, notably 403 Forbidden errors likely due to bot detection mechanisms. Although we used Puppeteer’s Extra Stealth plugin along with valid user-agent headers to minimize detection, we speculate that our IP addresses might have been flagged by advanced bot detection systems. To mitigate this in future work, researchers may employ proxies or VPNs to reduce detection and more comprehensively measure script deployment [12].

Companies like DataDome and PerimeterX/Human Security present unique challenges since they offer multiple products, including behavioral biometric solutions. Our methodology carefully distinguished scripts primarily based on the intensity of behavioral data collection, focusing only on scripts engaged in behavioral biometrics. Furthermore, we

identified interesting cases such as `antifrod.js` and `anti-bot.js` scripts on bank login pages. These scripts collect significant behavioral data, yet due to our inability to confidently associate them with known companies, we conservatively excluded them from our analysis.

Industry-specific trends emerged clearly during our analysis. For example, GroupIB frequently appears in gambling sites, Sardine in fintech and investment platforms, BioCatch in banking, and Forter primarily in e-commerce. F6 predominantly serves Russian websites, and Iovation has a broad presence across dating, adult websites, and other sectors. Such industry-specific patterns underline the diverse and targeted nature of behavioral biometric deployment.

We also uncovered specific patterns of deployment through authentication providers such as Fiserv and FIS Global, prevalent in the U.S. banking sector. Websites using Fiserv’s authentication portal predominantly incorporated Iovation scripts, while FIS Global authentication pages integrated BehavioSec scripts. NuData exhibited similar patterns across various banking websites, often sharing identical user interface designs (Figure 6.2).

Our methodology deliberately erred on the side of conservatism to avoid falsely attributing scripts to behavioral biometric vendors. Consequently, our reported prevalence likely represents a lower-bound estimate.

Interestingly, commonly used ad blockers and privacy-focused browser extensions (e.g., Privacy Badger [45]) utilizing lists such as EasyPrivacy often block behavioral biometric service providers’ domains. However, we observed minimal impact on site functionality when these scripts were blocked. This raises questions about the necessity of these scripts from a user-experience perspective.

Furthermore, we noticed demographic disparities in the deployment of these scripts, where certain companies activated behavioral biometric scripts for users in specific regions (e.g., U.S. versus Canada). While this may simply reflect phased rollouts, it nonetheless suggests potential disparities in user account protections across regions, an issue warranting future investigation.

In evaluating our Login Crawler against prior work, we observed that it outperformed comparable crawlers such as SSO-Monitor [83] and Double-Edge Sword [144]. Our crawler successfully discovered login pages for 53.3% of the top 100K websites, translating to an effective discovery rate of at least **46.9%** after adjusting for precision. This represents a significant improvement over Double-Edge Sword’s **27.05%** success rate and SSO-Monitor’s **4.6%** on the top 1 million Tranco sites.

Newly identified behavioral biometric vendors such as JuicyScore utilize cognitive behavioral patterns, analyzing user vocabulary, sentence structure, and navigation habits to

augment security measures. Similarly, companies like NEC are exploring advanced biometric modalities, such as vein-based and ear-acoustic authentication, highlighting the evolving landscape and the expanding definition of behavioral biometrics [87, 115].

The deployment of multiple behavioral biometric scripts by a single domain (e.g., TSB Bank, Wells Fargo, and Citi.com using BehavioSec alongside BioCatch) remains uncommon but noteworthy. The rationale for this redundancy remains unclear and warrants further investigation into the interplay and effectiveness of multiple concurrent behavioral biometric solutions.

Scripts from companies like Darwinium demonstrate deliberate stealth tactics, loading resources via first-party domains to evade ad-blockers and privacy tools. This method successfully circumvents domain blocklists, complicating mitigation efforts by privacy-conscious users and highlighting the ongoing arms race between user privacy technologies and data collection methods [33].

Variations in script deployment even within single entities were noted, such as specific login pages within a domain employing behavioral biometrics while others remained unprotected. Notably, Hancock Whitney Bank and StoneBank illustrate such inconsistencies, further emphasizing the nuanced and complex deployment strategies by financial institutions. As banks have multiple types of accounts they will have multiple login pages. For these two banks, we found some login pages to be protected by behavioral biometric providers while others login pages were not.

Browser-specific script failures, particularly evident between Chrome and Firefox, were also observed. Scripts relying on APIs disabled or altered in Firefox (e.g., BatteryManager API) may fail to function consistently, potentially weakening account protection against attackers using unsupported browsers. The impacts of browser fingerprinting countermeasures on behavioral biometric systems' accuracy are thus an essential area for future study since these scripts rely on browser fingerprints to complement a user's behavioral pattern.

Finally, despite extensive efforts, we cannot fully validate the backend processing of collected behavioral data—a limitation common in web measurement studies. While scripts clearly engage in behavioral data collection, without direct insights into backend analytics processes, we must acknowledge uncertainty regarding exact data usage. Nevertheless, through careful association with known vendors and vendor communications, we have minimized these ambiguities as to the best of our ability.

Overall, our findings provide a comprehensive overview of behavioral biometric script prevalence, deployment patterns, and potential limitations, laying the groundwork for further exploration into this rapidly evolving field.

6.5 Conclusion

This chapter presented the first large-scale measurement study of behavioral biometric deployment on the web, examining both the U.S. banking sector and the CrUX top 100K websites. Through the development of an automated machine learning classifier and comprehensive crawling infrastructure, we systematically identified and analyzed behavioral biometric script prevalence across diverse web environments.

Our key finding reveals a clear security-driven deployment pattern: behavioral biometrics appear on **15.8%** of banking login pages versus **6.1%** of banking landing pages, and **1.79%** of general web login pages compared to **0.506%** of landing pages. This demonstrates that organizations strategically deploy these technologies where they are most needed—protecting high-risk authentication interfaces rather than general browsing experiences.

Beyond quantifying prevalence, our study uncovered the operational landscape of behavioral biometrics, including industry-specific deployment patterns, authentication provider integrations, and evasion techniques such as first-party script hosting. The successful identification of five previously unknown vendors validates our vendor-agnostic machine learning approach and suggests the actual prevalence may exceed our conservative estimates.

These findings establish behavioral biometrics as a significant component of modern web security infrastructure, particularly in financial services, while highlighting important research directions including privacy implications, adversarial robustness, and cross-browser compatibility. This work provides the foundational measurement framework necessary for future studies examining the evolving role of behavioral biometrics in web security.

Table 6.9: Behavioral Biometric Vendors Identified via Login Pages (N = 53,308 domains)

Vendor	Number of Domains	Example Domains
Forter	424	insta360.com, aritzia.com
Cheq	286	stripe-club.com, hrblock.com
Iovation	267	stake.bet, poshmark.com, asiandating.com
BehavioSec	198	truist.com, sky.com, bestbuy.com
PerimeterX	134	stockx.com, fanduel.com, chegg.com
BioCatch	113	scotiabank.com, hdfcbank.com
Nudata	66	chime.com, ticketmaster.com
ArkoseLabs	57	coursera.org, postmates.com
PingOne	26	delta.com, screwfix.com
TransmitSecurity	19	interac.ca, rogers.com
GroupIB	18	starcasino.it, sports.ru
Accertify	12	seamless.com, starbucks.com
Sardine	10	wealthsimple.com, kraken.com
Yofi	10	shipstation.com
DataDome	9	paypal.com
Darwinium	7	udemy.com
DataVisor	5	booking.com
F5 Risk Services	5	klickbca.com
ThreatMark	5	erstebank.hr
Feedzai	4	tatrabank.sk
F6	2	banki.ru
Verisoul.ai	2	coinpayu.com
GeeTest	1	w88halong.com
JuicyScore	1	cash-express.ph

Chapter 7

Conclusion

7.1 Key Contributions

This thesis systematically explores the landscape and implications of behavioral biometric technologies on the modern web, employing a comprehensive approach combining manual analysis, automated static analysis, advanced web crawling techniques, and machine learning classification methods. Key contributions of this work include:

- Identification and detailed characterization of behavioral biometric scripts from major providers, including BioCatch, BehavioSec, TransUnion’s Iovation, and Mastercard’s NuData, among others.
- Development and deployment of LoginGPT, an advanced web crawler enhanced by Large Language Models (LLMs), significantly outperforming existing solutions in accurately detecting login pages across a diverse set of websites.
- Introduction of a hybrid static-dynamic taint analysis framework leveraging Visible V8, providing robust and precise tracing of behavioral biometric data flows within JavaScript, crucial for automated script detection and characterization.
- Implementation of a supervised machine learning approach employing Random Forest classifiers to accurately identify and classify behavioral biometric scripts based on vendor-agnostic features derived from detailed static taint analysis.

- Conducting large-scale empirical studies on the prevalence and distribution of behavioral biometric scripts across 9,502 U.S. banking websites and the top 100,000 domains from the Chrome User Experience (CrUX) dataset, providing a comprehensive view of their deployment.
- Discovery of five previously unknown behavioral biometric providers, expanding current knowledge about the behavioral biometric landscape.

7.2 Important Findings

Our extensive evaluation yielded several important findings (addressing the research goals mentioned in Chapter 1):

- **Quantified Prevalence:** We measured behavioral biometric adoption across 9,502 U.S. banks and 100K popular websites, finding 15.8% of banking domains with discoverable login pages and 1.79% of general web domains with discoverable login pages use these technologies; confirming their strategic role in protecting high-risk interactions.
- **Market Leaders and Adoption Patterns:** Among the vendors identified, Forter leads the general web login study with 424 domains, while in banking, Iovation dominates with 595 domains followed by BehavioSec with 113 domains. This reveals clear sector preferences: financial institutions favor established security providers like Iovation, while general web platforms choose fraud prevention specialists like Forter.
- **Login Pages Are Primary Targets:** Our hypothesis proved correct—domains with behavioral biometric scripts on login pages consistently outnumber those with scripts on landing pages (15.8% vs 6.1% for banking, 1.79% vs 0.51% for general web), demonstrating that organizations deploy these technologies specifically to protect authentication workflows.
- **Privacy-Security Tensions:** Vendors show varied approaches to data collection and obfuscation, creating a sophisticated but opaque industry that raises questions about user awareness and consent in security deployments.

7.3 Future Work

Based on our findings and identified limitations, several avenues for future research emerge:

- Expanding the static taint analysis framework to include dynamic taint tracking in real-time scenarios could further enhance the accuracy and robustness of behavioral biometric detection.
- Extending the LoginGPT crawler to handle more complex web interaction scenarios, such as multi-step authentication flows, single sign-on (SSO) integration points, and dynamic content loading.
- Investigating user perception and understanding of behavioral biometric technologies to better gauge privacy concerns and inform regulatory frameworks.
- Exploring adversarial scenarios where behavioral biometric data could potentially be exploited for malicious purposes, emphasizing the need for stronger privacy-preserving mechanisms within these scripts.
- Conducting cross-regional comparative studies to assess differences in behavioral biometric adoption, compliance with privacy regulations (e.g., GDPR, CCPA), and impact on user privacy and security across different jurisdictions.
- Enhancing machine learning models with advanced feature engineering methods, such as incorporating natural language processing (NLP) techniques to better understand and classify obfuscated or dynamically generated behavioral biometric scripts.

In conclusion, this thesis provides a comprehensive analysis and robust framework for detecting, characterizing, and measuring behavioral biometric technologies on the web, offering valuable insights for researchers, industry professionals, and policymakers concerned with digital identity management and privacy protection.

References

- [1] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. The web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 674–689, 2014.
- [2] Gunes Acar, Marc Juarez, Nick Nikiforakis, Claudia Diaz, Seda Gürses, Frank Piessens, and Bart Preneel. Fpdetective: dusting the web for fingerprinters. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1129–1140. ACM, 2013.
- [3] Accertify Inc. Accertify – digital risk management and fraud prevention solutions. <https://www.accertify.com/>, 2025. Accessed: 2025-06-01.
- [4] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in neural information processing systems*, 35:23716–23736, 2022.
- [5] Nabil Almashfi and Lunjin Lu. Static taint analysis for javascript programs. In *International Conference on Tools and Methods for Program Analysis*, pages 155–167. Springer, 2019.
- [6] Arkose Labs. Arkose labs: Securing online transactions and mitigating bots, 2025. Accessed: 2025-06-05.
- [7] Aswad546. Fingerprinting apis list, 2024. GitHub repository.
- [8] aswad546. behavioral-biometrics-scripts. GitHub repository, July 2025. <https://github.com/aswad546/behavioral-biometrics-scripts>.

- [9] aswad546. `company_clients.json`: List of company clients from behavioralbiometricsa repository. GitHub repository, June 2025. Accessed: 2025-06-19.
- [10] Pouneh Nikkhah Bahrami, Umar Iqbal, and Zubair Shafiq. Fp-radar: Longitudinal measurement and early detection of browser fingerprinting. *arXiv preprint arXiv:2112.01662*, 2021.
- [11] Alexandru Bara. Finding behavioural biometrics scripts on the web using dynamic taint analysis. 2025.
- [12] berstend. `puppeteer-extra-plugin-stealth`, 2023. Version 2.11.2. Accessed April 28, 2025. A Puppeteer plugin that applies various techniques to make detection of headless Puppeteer harder.
- [13] BioCatch. Biocatch expands digital identity position with aimbrain acquisition, February 2020. Accessed: 2025-06-05.
- [14] BioCatch. Biocatch strengthens collaboration with microsoft cloud for financial services. [BioCatch Press Release](#), June 2023. Accessed: 2025-06-05.
- [15] BioCatch. Biocatch – behavioral biometrics to prevent fraud & build trust. <https://www.biocatch.com/>, 2025. Accessed: 2025-06-01.
- [16] BioCatch. Company overview. <https://www.biocatch.com/hubfs/FACTSHEET.pdf?t=1498242263377>, n.d. Accessed: 2025-06-05.
- [17] Biometric Update. Biocatch patents touchscreen pressure measuring method to extend behavioral biometric innovation. *Biometric Update*, March 2019. Accessed: 2025-06-01.
- [18] Ahmed Bouhoula, Karel Kubicek, Amit Zac, Carlos Cotrini, and David Basin. Automated large-scale analysis of cookie notice compliance. In *Proc. 33rd USENIX Security Symposium*, 2024.
- [19] Soumaya Boussaha, Lukas Hock, Miguel Bermejo, Rubén Cuevas Rumin, Angel Cuevas Rumin, David Klein, Martin Johns, Luca Compagna, Daniele Antonioli, and Thomas Barber. Fp-tracer: Fine-grained browser fingerprinting detection via taint-tracking and entropy-based thresholds. *Proceedings on Privacy Enhancing Technologies*, 2024.
- [20] ByteDance Ltd. Bytedance official website. <https://www.bytedance.com>, 2025. Accessed July 2025.

- [21] Callsign Inc. Callsign – digital identity, authentication, and fraud prevention. <https://www.callsign.com/>, 2025. Accessed: 2025-06-01.
- [22] Diane Carlson. The future of mobile banking security is silent and invisible. <https://www.samsungsds.com/eu/blog/future-mobile-banking.html>, October 2017. Accessed: 2025-06-05.
- [23] Michelle Castell. Mitigating online account takeovers: The case for education. Technical report, Federal Reserve Bank of Atlanta, 2013.
- [24] Celebris Technologies plc. About celebris. <https://www.celebris.com/about-us>, 2025. Accessed: 2025-06-05.
- [25] Celebris Technologies plc. Digital data analytics made simple. <https://www.celebris.com/data-platform/digital-analytics>, 2025. Accessed: 2025-06-05.
- [26] Zhe Chen, Weiyun Wang, Hao Tian, Shenglong Ye, Zhangwei Gao, Erfei Cui, Wenwen Tong, Kongzhi Hu, Jiapeng Luo, Zheng Ma, et al. How far are we to gpt-4v? closing the gap to commercial multimodal models with open-source suites. *Science China Information Sciences*, 67(12):220101, 2024.
- [27] Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. SeeClick: Harnessing gui grounding for advanced visual gui agents. *arXiv preprint arXiv:2401.10935*, 2024.
- [28] CHEQ. Browser fingerprinting for fraud detection. <https://cheq.ai/browser-fingerprinting-fraud-detection/>, 2023. Accessed: 2025-05-15.
- [29] Cloudflare, Inc. Ai labyrinth, 2025.
- [30] Contentsquare. Clicktale is now contentsquare, 2025. Accessed: 2025-06-05.
- [31] Stacy Cowley. Behavioral biometrics: Banks turn to body movements to fight fraud. *The New York Times*, August 13 2018. Accessed: 2025-06-01.
- [32] D4t4 Solutions plc. Final results for the year ended 31 march 2023. Technical report, D4t4 Solutions plc, July 2023. Accessed: 2025-06-05.
- [33] Darwinium. Tags deployment - why aps.yourhost.com? <https://docs.darwinium.com/docs/tags-deployment>, 2024. Accessed: 2025-07-03.

- [34] DataDome. How datadome uses fingerprinting to block bot attacks. <https://datadome.co/bot-management-protection/fingerprinting-for-bot-detection/>, 2023. Accessed: 2025-05-15.
- [35] DataDome. Datadome bot protection and online fraud prevention. <https://datadome.co/>, 2025. Accessed: 2025-06-01.
- [36] DataDome. Datadome: Cyberfraud and bot protection solution, 2025. Accessed: 2025-06-05.
- [37] Michalis Diamantaris, Francesco Marcantoni, Sotiris Ioannidis, and Jason Polakis. The seven deadly sins of the html5 webapi: A large-scale study on the risks of mobile sensor-based attacks. *ACM Transactions on Privacy and Security*, 23(4), July 2020.
- [38] Kostas Drakonakis, Sotiris Ioannidis, and Jason Polakis. The cookie hunter: Automated black-box auditing for web authentication and authorization flaws. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1953–1970, 2020.
- [39] DuckDuckGo. Tracker radar collector. <https://github.com/duckduckgo/tracker-radar-collector>, 2025. Accessed: 2025-06-06.
- [40] Antonin Durey, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. Fp-redemption: Studying browser fingerprinting adoption for the sake of web security. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 18th International Conference, DIMVA 2021, Virtual Event, July 14–16, 2021, Proceedings 18*, pages 237–257. Springer, 2021.
- [41] Zakir Durumeric and David Adrian. Chrome (crux) top million websites, 12 2022. Downloadable snapshots of the Chrome Top Million Websites pulled from public CrUX data in Google BigQuery.
- [42] Dynatrace LLC. Dynatrace: Observability and performance monitoring. <https://www.dynatrace.com>, 2025. Accessed July 2025.
- [43] EasyList Community. Easyprivacy: Subscription for blocking tracking and behavioral advertising. <https://easylist.to/>, 2024. Accessed: 2025-07-03.
- [44] Homam Reda El-Taj, Danah Hamedah, and Rawan Saeed. Artificial intelligence and advanced cybersecurity to mitigate credential-stuffing attacks in the banking industry. *International Journal of Computational and Experimental Science and Engineering*, 2025.

- [45] Electronic Frontier Foundation. Privacy badger. <https://privacybadger.org/>, 2024. Accessed: 2025-07-03.
- [46] Steven Englehardt and Arvind Narayanan. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 1388–1401, 2016.
- [47] Experian. Liminal link index: Account takeover prevention in banking. Technical report, Experian, October 2024. Accessed: 2025-06-05.
- [48] Experian. Experian corporate website, 2025. Accessed: 2025-06-05.
- [49] F6.ru. F6: Fraud prevention and behavioral biometrics. <https://www.f6.ru/products>, 2025. Accessed July 2025.
- [50] Ahmad FaizKhademi, Mohammad Zulkernine, and Komminist Weldemariam. Fp-guard: Detection and prevention of browser fingerprinting. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 293–308. Springer, 2015.
- [51] Aurore Fass, Michael Backes, and Ben Stock. Jstap: A static pre-filter for malicious javascript detection. In *Proceedings of the 35th Annual Computer Security Applications Conference*, pages 257–269, 2019.
- [52] Federal Deposit Insurance Corporation. Bankfind suite, 2025. Accessed April 28, 2025. Provides data on FDIC-insured banks and their locations.
- [53] Federal Deposit Insurance Corporation. Bankfind suite: Institution directory. <https://banks.data.fdic.gov/bankfind-suite/bankfind>, n.d. Accessed: 2025-07-01.
- [54] Federal Deposit Insurance Corporation. Federal deposit insurance corporation (fdic). <https://www.fdic.gov/>, n.d. Accessed: 2025-07-01.
- [55] Feedzai. Feedzai: Financial fraud detection software. <https://www.feedzai.com>, 2025. Accessed: 2025-06-06.
- [56] Jeanne Ferrante, Karl J Ottenstein, and Joe D Warren. The program dependence graph and its use in optimization. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 9(3):319–349, 1987.
- [57] Forter. The forter trust platform for digital commerce. <https://www.forter.com/platform/>, 2025. Accessed: 2025-06-06.

- [58] Imane Fouad, Cristiana Santos, Arnaud Legout, and Nataliia Bielova. Detecting, measurement, and lawfulness of cookie respawning with browser and machine fingerprinting. In *Proceedings on Privacy Enhancing Technologies (PETS)*, pages 63–83, 2022. Defines stateful tracking via cookies storing a unique user identifier in client-side storage.
- [59] Fraud.com. Browser fingerprinting: A thorough overview. <https://fraud.com/blog/browser-fingerprinting/>, 2023. Accessed: 2025-05-15.
- [60] Fullstory, Inc. Fullstory: Surface user sentiment with behavioral data, 2025. Accessed: 2025-06-05.
- [61] GeeTest. Geetest captcha and behavioral biometrics. <https://www.geetest.com>, 2025. Accessed July 2025.
- [62] Mohammad Ghasemisharif, Chris Kanich, and Jason Polakis. Towards automated auditing for account and session management flaws in single sign-on deployments. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1774–1790. IEEE, 2022.
- [63] Google. *Chrome DevTools*, 2024. Accessed: 2025-06-05.
- [64] Google. Puppeteer: Node.js library for browser automation. <https://pptr.dev/>, 2024. Accessed: 2025-05-12.
- [65] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The L^AT_EX Companion*. Addison-Wesley, Reading, Massachusetts, 1994.
- [66] Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. Navigating the digital world as humans do: Universal visual grounding for gui agents. *arXiv preprint arXiv:2410.05243*, 2024.
- [67] Graboxy. Biometric authentication part ii: Behavioral biometrics. <https://graboxy.com/post/1818/biometric-authentication-part-ii-behavioral-biometrics-2>, 2024. Accessed: 2025-06-05.
- [68] Grafana Labs. Grafana: The open observability platform. <https://grafana.com/>, n.d. Accessed: 2025-07-01.
- [69] Grafana Labs. Loki: Like prometheus, but for logs. <https://grafana.com/oss/loki/>, n.d. Accessed: 2025-07-01.

- [70] Grand View Research. Behavioral biometrics market size, share & trends analysis report by component, by type, by application, by deployment, by enterprise size, by end-use, by region, and segment forecasts, 2023 - 2030. <https://www.grandviewresearch.com/industry-analysis/behavioral-biometrics-market>, 2023. Accessed: 2025-06-01.
- [71] Sjors Haanen, Tim van Zalingen, Aidan Barrington, and Ruben de Vries. Detection of browser fingerprinting by static javascript code classification. 2018. University of Amsterdam.
- [72] Adrian Herrera. Optimizing away javascript obfuscation. *arXiv preprint arXiv:2009.09170*, 2020.
- [73] Susan Horwitz, Thomas Reps, and David Binkley. Interprocedural slicing using dependence graphs. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 12(1):26–60, 1990.
- [74] Hotjar Ltd. Hotjar: Website heatmaps & behavior analytics tools, 2025. Accessed: 2025-06-05.
- [75] HUMAN Security. Human security: Protecting the digital experience, 2025. Accessed: 2025-06-05.
- [76] Human Security Inc. Human security (formerly perimeterx): Bot and fraud mitigation. <https://www.humansecurity.com/>, 2025. Accessed: 2025-06-01.
- [77] IMARC Group. Behavioral biometrics market: Global industry trends, share, size, growth, opportunity and forecast 2025-2033. <https://www.imarcgroup.com/behavioral-biometrics-market>, 2025. Accessed: 2025-06-01.
- [78] Internet Archive. Wayback machine, 2025. Accessed: 2025-06-05.
- [79] iovation Inc. iovation – device-based fraud prevention and authentication. <https://www.iovation.com/>, 2025. Accessed: 2025-06-01.
- [80] Umar Iqbal, Steven Englehardt, and Zubair Shafiq. Fingerprinting the fingerprinters: Learning to detect browser fingerprinting behaviors. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1143–1161. IEEE, 2021.
- [81] Umar Iqbal, Steven Englehardt, and Zubair Shafiq. Fingerprinting the fingerprinters: Learning to detect browser fingerprinting behaviors. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1143–1161. IEEE, 2021.

- [82] Irdeto. Solutions for the video game industry. <https://irdeto.com/video-games>, 2025. Accessed: 2025-06-05.
- [83] Louis Jannett, Christian Mainka, Maximilian Westers, Andreas Mayer, Tobias Wich, and Vladislav Mladenov. Sok: Sso-monitor-the current state and future research directions in single sign-on security measurements. In *2024 IEEE 9th European Symposium on Security and Privacy (EuroSP)*, pages 173–192. IEEE, 2024.
- [84] Nidhal Jegham, Marwan Abdelatti, and Abdeltawab Hendawi. Visual reasoning evaluation of grok, deepseek janus, gemini, qwen, mistral, and chatgpt. *arXiv preprint arXiv:2502.16428*, 2025.
- [85] Hugo Jonker, Stefan Karsch, Benjamin Krumnow, and Marc Slegers. Shepherd: a generic approach to automating website login. 2020.
- [86] Jordan Jueckstock and Alexandros Kapravelos. Visiblev8: In-browser monitoring of javascript in the wild. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19)*, pages 109–123. Association for Computing Machinery, 2019.
- [87] JuicyScore. Introduction to behavioral biometrics. <https://juicycore.ai/en/resource/introduction-to-behavioral-biometrics>, 2024. Accessed: 2025-07-03.
- [88] JuicyScore LLC. Juicycore: Advanced digital risk assessment. <https://juicycore.ai>, 2025. Accessed July 2025.
- [89] Ariya Kantor. Esprima: EcmaScript parsing infrastructure for multipurpose analysis. <https://esprima.org>, 2025. Accessed June 12, 2025.
- [90] Rahul Kanyal and Smruti R Sarangi. Panoptichrome: A modern in-browser taint analysis framework. In *Proceedings of the ACM Web Conference 2024*, pages 1914–1922, 2024.
- [91] Donald Knuth. *The T_EXbook*. Addison-Wesley, Reading, Massachusetts, 1986.
- [92] Kount. Kount: Fraud detection and chargeback management solutions, 2025. Accessed: 2025-06-05.
- [93] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management

- for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.
- [94] Leslie Lamport. *LaTeX — A Document Preparation System*. Addison-Wesley, Reading, Massachusetts, second edition, 1994.
 - [95] Pierre Laperdrix, Nataliaia Bielova, Benoit Baudry, and Gildas Avoine. Browser fingerprinting: A survey. *ACM Transactions on the Web (TWEB)*, 14(2):1–33, 2020.
 - [96] Pierrick Laperdrix et al. Detecting browser extensions via injected style sheets. In *USENIX Security Symposium*, 2021. Notes that stateful tracking uses cookies and other stateful identifiers.
 - [97] LastPass. Lastpass browser extension, 2025. Version 4.x, Accessed April 21, 2025.
 - [98] Alexander Lawall. Fingerprinting and tracing shadows: The development and impact of browser fingerprinting on digital privacy. *arXiv preprint arXiv:2411.12045*, 2024.
 - [99] Xuanyu Lei, Zonghan Yang, Xinrui Chen, Peng Li, and Yang Liu. Scaffolding coordinates to promote vision-language coordination in large multi-modal models. *arXiv preprint arXiv:2402.12058*, 2024.
 - [100] LexisNexis Risk Solutions. Lexisnexis risk solutions acquires behavioral biometric innovator behaviosec. [LexisNexis Risk Solutions Press Room](#), May 2022. Accessed: 2025-06-05.
 - [101] LexisNexis Risk Solutions. Behaviosec – a behavioral biometrics solution. <https://risk.lexisnexis.com/products/behaviosec>, 2025. Accessed: 2025-06-01.
 - [102] LexisNexis Risk Solutions. First-party fraud surpasses scams to become the leading form of fraud; account takeover fraud remains a major threat. Cybercrime report, LexisNexis Risk Solutions, Atlanta, GA, May 2025. Analysis of over 104 billion transactions through the LexisNexis Digital Identity Network, highlighting ATO as 27% of fraud and over three billion automated takeover attempts. [Available online](#).
 - [103] Hong Li, Nanxi Li, Yuanjie Chen, Jianbin Zhu, Qinlu Guo, Cewu Lu, and Yong-Lu Li. The labyrinth of links: Navigating the associative maze of multi-modal llms. *arXiv preprint arXiv:2410.01417*, 2024.

- [104] Tianyi Li, Xiaofeng Zheng, Kaiwen Shen, and Xinhui Han. Fpflow: detect and prevent browser fingerprinting with dynamic taint analysis. In *China Cyber Security Annual Conference*, pages 51–67. Springer Nature Singapore Singapore, 2021.
- [105] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *NeurIPS*, 2023.
- [106] Zengrui Liu, Jimmy Dani, Yinzhi Cao, Shujiang Wu, and Nitesh Saxena. The first early evidence of the use of browser fingerprinting for online tracking. In *Proceedings of the ACM on Web Conference 2025*, pages 4980–4995, 2025.
- [107] Ann Longmore-Etheridge. Deal fraud out. *Security Management*, September 2005. Accessed: 2025-06-05.
- [108] Francesco Marcantoni, Michalis Diamantaris, Sotiris Ioannidis, and Jason Polakis. A large-scale study on the risks of the html5 webapi for mobile sensor-based attacks. In *Proceedings of the 30th International World Wide Web Conference (WWW '19)*. ACM, 2019.
- [109] Mastercard. Mastercard enhances security of the internet of things with the acquisition of nudata security inc. [Mastercard Investor News](#), March 2017. Accessed: 2025-06-05.
- [110] MCP Insight Ltd. Mcp shield: Real-time behavioral fraud detection for mobile/web transactions. <https://mcpinsight.com/mcp-shield/>, 2025. Accessed July 2025.
- [111] Medallia, Inc. Medallia: Customer experience management. <https://www.medallia.com>, 2025. Accessed July 2025.
- [112] Microsoft Corporation. Microsoft clarity, 2025. Accessed: 2025-06-05.
- [113] Mordor Intelligence. Behavioral biometrics market - growth, trends, covid-19 impact, and forecasts (2025 - 2030). <https://www.mordorintelligence.com/industry-reports/behavioral-biometrics-market>, 2025. Accessed: 2025-06-01.
- [114] Meenatchi Sundaram Muthu Selva Annamalai, Emiliano De Cristofaro, and Igor Bilogrevic. Beyond the crawl: Unmasking browser fingerprinting in real user interactions. In *Proceedings of the ACM on Web Conference 2025*, pages 3896–3907, 2025.

- [115] NEC Corporation. Biometrics: Enhancing identity with facial, fingerprint, iris, vein and acoustic authentication. <https://www.nec.com/en/global/solutions/biometrics/index.html>, 2024. Accessed: 2025-07-03.
- [116] NerdyData Inc. Nerdydata – source code search engine. <https://www.nerdydata.com/>, 2025. Accessed: 2025-06-01.
- [117] Nick Nikiforakis, Wouter Joosen, and Benjamin Livshits. Privaricator: Deceiving fingerprinters with little white lies. In *Proceedings of the 24th International Conference on World Wide Web*, pages 820–830, 2015.
- [118] NuData Security. Nudata security – passive behavioral biometrics and fraud prevention. <https://nudatasecurity.com/>, 2025. Accessed: 2025-06-01.
- [119] Authors of JSimpo. Jsimpo: Structural deobfuscation of javascript programs. In *Proceedings of the [Conference Name]*. ACM, 2025.
- [120] Will Peakin. Royal bank of scotland widens use of behavioural biometrics to combat fraud. *Futurescot*, August 14 2018. Accessed: 2025-06-01.
- [121] Permira. Permira completes acquisition of majority position in biocatch at \$1.3 billion valuation. *Permira News & Insights*, September 9 2024. Accessed: 2025-06-01.
- [122] Permira. Permira completes acquisition of majority position in biocatch at \$1.3 billion valuation. [Permira News & Insights](#), September 2024. Accessed: 2025-06-05.
- [123] Tien-Huy Pham, Quoc-Huy Vo, Ha Dao, and Kensuke Fukuda. Ssologin: A framework for automated web privacy measurement with sso logins. In *Proceedings of the 18th Asian Internet Engineering Conference*, pages 69–77, 2023.
- [124] Ping Identity. Behaviosec behavioral biometrics integration. <https://support.pingidentity.com/s/marketplace-integration/a7i1W000000006UhQAI/behaviosec-behavioral-biometrics>, 2020. Accessed: 2025-06-05.
- [125] Ping Identity. Ping identity announces the acquisition of securedtouch to accelerate identity fraud capabilities. [Ping Identity Press Release](#), June 2021. Accessed: 2025-06-05.
- [126] Ping Identity. The pingone cloud platform. <https://videos.pingidentity.com/detail/video/6292567684001/the-pingone-cloud-platform>, 2021. Accessed: 2025-06-05.

- [127] Ping Identity. Securedtouch. <https://support.pingidentity.com/s/directory-profile/a7h1W000000C10oQAC/securedtouch>, 2025. Accessed: 2025-06-05.
- [128] Plurilock Security Inc. Plurilock defend™ continuous zero trust authentication. <https://plurilock.com/products/defend/>, 2025. Accessed: 2025-06-05.
- [129] Chris Preimesberger. Iovation security: Product overview and insight. *eWEEK*, December 2018. Accessed: 2025-06-05.
- [130] Dmytro Progonov, Valentyna Cherniakova, Pavlo Kolesnichenko, and Andriy Oliynyk. Behavior-based user authentication on mobile devices in various usage contexts. *EURASIP Journal on Information Security*, 2022(1):6, 2022.
- [131] Prometheus Authors. Prometheus: Monitoring system and time series database. <https://prometheus.io/>, n.d. Accessed: 2025-07-01.
- [132] Proofpoint. What is account takeover fraud? how browser fingerprinting helps prevent it. <https://www.proofpoint.com/us/threat-reference/account-takeover-fraud>, 2023. Accessed: 2025-05-15.
- [133] Prove. Prove to acquire mobile behavioral biometrics company unifyid to accelerate dynamic multi-factor authentication. <https://www.prove.com/blog/prove-acquires-unifyid>, June 2021. Accessed: 2025-06-05.
- [134] PYMNTS. Lexisnexis risk solutions buys behaviosec for behavioral biometrics. *PYMNTS*, May 2022. Accessed: 2025-06-05.
- [135] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.
- [136] Research and Markets. Behavioral biometrics market by authentication type, solution component, deployment mode, end-user industry, application - global forecast to 2030. <https://www.researchandmarkets.com/report/behavioral-characteristics>, 2025. Accessed: 2025-06-01.
- [137] Suood Al Roomi and Frank Li. A large-scale measurement of website login policies. In *Proc. 32nd USENIX Security Symposium*, 2023.
- [138] Kimberly Ruth, Deepak Kumar, Brandon Wang, Luke Valenta, and Zakir Durumeric. Toppling top lists: Evaluating the accuracy of popular website lists. In *Proc. ACM Internet Measurement Conference (IMC)*, 2022.

- [139] Samsung SDS. Samsung sds and biocatch join forces to enable secure, frictionless mobile experiences. <https://www.samsungsds.com/us/news/pr-biocatch-samsung.html>, October 2017. Accessed: 2025-06-05.
- [140] SearXNG Contributors. Searxng: Privacy-respecting, hackable metasearch engine, 2023. Accessed: 2025-04-28.
- [141] SecuredTouch. Securedtouch – behavioral biometrics and bot detection. <https://www.securedtouch.com/>, 2025. Accessed: 2025-06-01.
- [142] Jayasree Sengupta, Tanya Shreedhar, Robert Kramer, and Vaibhav Bajpai. Through the lens of google crux: Dissecting web browsing experience across devices and countries. In *Proc. IFIP Networking*, 2024.
- [143] Asuman Senol and Gunes Acar. Unveiling the impact of user-agent reduction and client hints: A measurement study. In *Proc. ACM Conference on Computer and Communications Security (CCS)*, 2023.
- [144] Asuman Senol, Alisha Ukani, Dylan Cutler, and Igor Bilogrevic. The double edged sword: identifying authentication pages and their fingerprinting behavior. In *Proceedings of the ACM Web Conference 2024*, pages 1690–1701, 2024.
- [145] Sentry, Inc. Sentry: Application monitoring and performance tracking. <https://sentry.io>, 2025. Accessed July 2025.
- [146] Andreea-Gabriela Sîrbu. Deobfuscating javascript code using character-based tokenization. *Studia Universitatis Babeş-Bolyai, Informatica*, 68(2):..., 2023.
- [147] Alexander Sjösten, Daniel Hedin, and Andrei Sabelfeld. Essentialfp: Exposing the essence of browser fingerprinting. In *2021 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 32–48. IEEE, 2021.
- [148] Aleksei Stafeev and Giancarlo Pellegrino. Sok: State of the krawlers – evaluating the effectiveness of crawling algorithms for web security measurements. In *Proc. 33rd USENIX Security Symposium*, 2024.
- [149] O Starov and N Nikiforakis. Tales of f s and caches: Persistent tracking in modern browsers. In *NDSS Symposium*, 2021. Describes stateful tracking via client-side storage that identifies returning users.

- [150] Straits Research. Behavioral biometrics market size, share & growth graph by 2033. <https://straitsresearch.com/report/behavioral-biometrics-market>, 2025. Accessed: 2025-06-01.
- [151] Junhua Su and Alexandros Kapravelos. Automatic discovery of emerging browser fingerprinting techniques. In *Proceedings of the ACM Web Conference 2023*, pages 2178–2188, 2023.
- [152] Junhua Su and Alexandros Kapravelos. Automatic discovery of emerging browser fingerprinting techniques. In *Proceedings of the ACM Web Conference 2023*, pages 2178–2188, 2023.
- [153] Aswad Tariq. List of 9502 u.s. banks compiled from community banker and fdic bankfind suite. [Google Drive](#), 2025. Compiled using data from http://www.thecommunitybanker.com/bank_links/ and <https://banks.data.fdic.gov/bankfind-suite/bankfind>. Accessed: 2025-07-01.
- [154] The Community Banker. Bank links directory, 2025. Accessed April 28, 2025. Directory of U.S. community banks and financial institutions.
- [155] The Community Banker. Us banks on the internet. http://www.thecommunitybanker.com/bank_links/, n.d. Accessed: 2025-07-01.
- [156] The Economist. Online identification is getting more and more intrusive. *The Economist*, May 23 2019. Accessed: 2025-06-01.
- [157] The Jaeger Authors. Jaeger: Open source distributed tracing. <https://www.jaegertracing.io/>, n.d. Accessed: 2025-07-01.
- [158] ThreatMark. Account takeover fraud: Everything you need to know for prevention, April 2025. Describes ThreatMark’s Behavioral Intelligence Platform using behavioral biometrics for real-time ATO detection in financial institutions. [Available online](#).
- [159] TransUnion. Transunion announces agreement to acquire iovation to strengthen fraud and identity solutions. [TransUnion Newsroom](#), May 2018. Accessed: 2025-06-05.
- [160] TypingDNA. Activelock: Continuous endpoint authentication using typing biometrics. <https://www.typingdna.com/activelock-continuous-authentication>, 2025. Accessed: 2025-06-05.

- [161] TypingDNA. Tutorials. <https://www.typingdna.com/docs/tutorials.html>, 2025. Accessed: 2025-06-05.
- [162] Steven Van Acker, Daniel Hausknecht, and Andrei Sabelfeld. Measuring login webpage security. In *Proceedings of the Symposium on Applied Computing*, pages 1753–1760, 2017.
- [163] VeriSoul.ai. Verisoul.ai: Behavioral biometric fraud detection. <https://verisoul.ai>, 2025. Accessed July 2025.
- [164] Versasec. Q&a with olov renberg, co-founder and coo of behaviosec. [Versasec Blog](#), May 2018. Accessed: 2025-06-05.
- [165] Victoria’s Secret. Victoria’s secret canada, 2025. Accessed: 2025-06-05.
- [166] P Wang, S Bai, S Tan, S Wang, Z Fan, J Bai, K Chen, X Liu, J Wang, W Ge, et al. Qwen2-vl: Enhancing vision-language model’s perception of the world at any resolution, 2024. URL <https://arxiv.org/abs/2409.12191>, 2024.
- [167] Fangyun Wei, Jinjing Zhao, Kun Yan, Hongyang Zhang, and Chang Xu. A large-scale human-centric benchmark for referring expression comprehension in the Imm era. *Advances in Neural Information Processing Systems*, 37:69566–69587, 2024.
- [168] WSPR Lab, North Carolina State University. VisibleV8 Crawler: Framework for large-scale crawling of urls with visiblev8. <https://github.com/wspr-ncsu/visiblev8-crawler>, 2025. GitHub repository, accessed June 12, 2025.
- [169] Shujiang Wu, Pengfei Sun, Yao Zhao, and Yinzhi Cao. Him of many faces: Characterizing billion-scale adversarial and benign browser fingerprints on commercial websites. In *NDSS*, 2023.
- [170] Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, et al. Os-atlas: A foundation action model for generalist gui agents. *arXiv preprint arXiv:2410.23218*, 2024.
- [171] Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. Aguviz: Unified pure vision agents for autonomous gui interaction. *arXiv preprint arXiv:2412.04454*, 2024.
- [172] Yandex LLC. Yandex: Web analytics and user behavior. <https://metrica.yandex.com>, 2025. Accessed July 2025.

- [173] Nicholas C. Zakas. espre: An esprima-compatible javascript parser built on acorn. <https://www.npmjs.com/package/espre>, 2025. Version 10.4.0, published June 2025, BSD-2-Clause license, accessed June 12, 2025.
- [174] Di Zhang, Junxian Li, Jingdi Lei, Xunzhi Wang, Yujie Liu, Zonglin Yang, Jiatong Li, Weida Wang, Suorong Yang, Jianbo Wu, et al. Critic-v: Vlm critics help catch vlm errors in multimodal reasoning. *arXiv preprint arXiv:2411.18203*, 2024.
- [175] Rui Zhao. Toward the flow-centric detection of browser fingerprinting. *Computers & Security*, 137:103642, 2024.
- [176] Zighra Inc. Zighra – ai-powered continuous authentication and threat detection. <https://zighra.com/>, 2025. Accessed: 2025-06-01.

APPENDICES

Appendix A

Login Crawler

A.1 Model Prompts

This appendix contains the prompts used with the OS-ATLAS model as referenced in Section [4.3.2](#).

A.1.1 Cookie Banner Detection Prompt

```
Analyze the provided image and determine if there are any visible popups or cookie
↪ banners.
If a popup is detected, where do I click to close it. Give me the coordinates of a
↪ cross icon in order to close it. If this does not exist give me the coordinates of
↪ the button inside the popup that exists.
If a cookie banner is detected return the position of the large Accept button inside a
↪ colored shape, for example oval or square.
If no popup or cookie banner exists Output: "No popups found".

Output Format:
Element Type: [Popup/Cookie Banner]
Description: [Brief description]
Bounding Box Coordinates: (x1, y1, x2, y2)

Guidelines:
- Only focus on popups or cookie banners.
- Provide precise bounding box coordinates.
```

Listing 1: Prompt for cookie banner detection using OS-ATLAS model

A.1.2 Login Element Detection Prompt

```
Analyze the provided image and identify where do I click to access the login page.
This may be an element labeled abstractly like Online Banking, My Account, Login or a
↪ person icon or even a form submit button associated with login credentials etc.

Output Format:
Element Type: Login Button
Description: [Brief description]
Bounding Box Coordinates: (x1, y1, x2, y2)

Guidelines:
- Only focus on the element that takes me to the login page.
- Provide precise bounding box coordinates.
```

Listing 2: Prompt for login element detection using OS-ATLAS model

A.1.3 Classification of Login Page Prompt

```
Analyze the provided image and determine if it contains input fields associated with
↳ the login flow of a web page. Specifically, look for:

Username or email input fields (e.g., forms with user ID, unique user ID, email
↳ address, or similar fields).
Password input fields (fields intended for password entry).
Follow this structured approach:

Identify all input fields in the image.
Filter out irrelevant input fields, such as those related to search, comments, or
↳ non-login-related data collection.
Determine if at least one relevant login-related input field is present and visible on
↳ the page.
Explain your reasoning step by step (Chain of Thought) to justify your decision.
Strictly output either "YES" or "NO" at the end, based on whether a login form
↳ containing at least one relevant input field is detected.
Output Format (Important):
After explaining your reasoning, respond strictly with either:

"YES" (if a relevant login input field is present and visible).
"NO" (if no relevant login input field is found).
```

Listing 3: Prompt for classification of login pages using Qwen2.5VL model

A.2 CrUX Top 1000 Domains Crawl Success

All figure properties can also be manipulated from the command line. Here's an example:

Table A.1: Domain Success and Failure Analysis of 1000 Crux Websites

Status	Count	Percentage (%)
Successful (Reachable)	900	90.0
Failures by Category:		
Access Denied/Restricted	51	5.1
403 Forbidden	49	4.9
429 Too Many Requests	1	0.1
451 Unavailable For Legal Reasons	1	0.1
Resource Not Found	12	1.2
404 Not Found	11	1.1
410 Gone	1	0.1
Timeout Issues	23	2.3
Process Timeouts	8	0.8
General Timeouts	15	1.5
Connection Issues	10	1.0
ERR_ABORTED (Frame Detached)	3	0.3
ERR_NAME_NOT_RESOLVED	7	0.7
Other Server Errors	4	0.4
490 Error	1	0.1
498 Error	1	0.1
503 Service Unavailable	1	0.1
526 Invalid SSL Certificate	1	0.1

Note: Percentages are calculated relative to the total 1000 websites.

Appendix B

Classifying Behavioral Biometric Scripts

B.1 Keywords For Known Companies

This section details known keywords that we use to identify scripts from behavioral biometric providers. Our table also includes keywords for companies whose scripts we did not find, these keywords stem from company white-papers about the product, API documentation and company domain names.

Table B.1: Security and Fraud Detection Companies and Associated Keywords

Company	Keywords
CallSign Inc	CallSignOne, callsign.com
Deepnet Security	deepnet, deepnetsecurity, typesense, nethone
MangoPay	mangopay.com, nethone.com
Mastercard Inc	nudatasecurity, unifyid, ndata, ndjsStaticVersion, nds.pmd
SecureAuth Corp	arculix, secureauth, acceptto, faye.acceptto, biobehavior
TypingDNA	typingdna, api.typingdna, getTypingPattern
BioCatch	self.cdApi, wupmessage, bcdn, we-stats
BehavioSec	behaviosec
Continued on next page	

Table B.1 – continued from previous page

Company	Keywords
ThreatMark	threatMark, javaScriptProbeURL
PingIdentity	pingOne, _pingOneSignalsToken
Bureau.id	bureau.id
Feedzai	feedzai, feedzai.cloud
Transmit Security	TSAccountProtectionProvider, TSAccountProtectionReady, tsPlatform, web.sdk.latest.js
Experian (Neuro-id)	nid.js, nid.id, data-nid-no-track, neuro-id, nid-adv
Cleafy	cleafy
Forter	forter.com, cdn4.forter
D4T4 Celebrus	celebrus
Arkose Labs	api.arkoselabs, arkoselabs, profilingInstance
Darwinium	geo_location_enabled, dwn_ns, dwn_add, dwn-profiling, dwn_
ThreatFabric	threatfabric
Accertify	accertify, prod.accdab
Human (PerimeterX)	pxAppId, PerimeterX, px-cloud
F5 Risk Services	riskservices
DataVisor	dv-api, datavisor
Graboxy	graboxy, GRABOXY-API-KEY
Sardine	sardine, d.dev.sardine.ai
DataDome	datadome
Iovation	Iesnare, wdp.js, Transunion, IGLOO
Yofi	yofiTelemetry, yofi-telemetry, yofi
Group IB	groupib
Untarget	untarget.ai, untarget
Cheq	cheqzone

B.2 Model Hyperparameters

B.2.1 Optimal Hyperparameters

The following tables present the optimal hyperparameters obtained through nested cross-validation for each model on both balanced and imbalanced datasets.

Optimal Hyperparameters for Imbalanced Dataset

Table B.2: Optimal Hyperparameters (Imbalanced Dataset)

Model	Optimal Hyperparameters
Random Forest	n_estimators=200 max_depth=10 min_samples_split=2 min_samples_leaf=1 max_features='sqrt' class_weight='balanced'
Gradient Boosting	n_estimators=200 max_depth=7 learning_rate=0.1 subsample=0.8 max_features='sqrt'
LightGBM	n_estimators=100 max_depth=6 learning_rate=0.2 num_leaves=15 min_child_samples=10 class_weight='balanced' feature_fraction=0.8 bagging_fraction=0.8 bagging_freq=0 min_gain_to_split=0.0
XGBoost	n_estimators=200 max_depth=3 learning_rate=0.2 subsample=0.9 colsample_bytree=1.0 reg_alpha=0 reg_lambda=0
SVM (RBF)	C=10

Continued on next page

Table B.2 – *Continued from previous page*

Model	Optimal Hyperparameters
	gamma='scale' kernel='rbf' class_weight='balanced' probability=True
Logistic Regression	C=10 class_weight='balanced' max_iter=1000 penalty='l1' solver='liblinear'
Naive Bayes	var_smoothing=1e-09

Optimal Hyperparameters for Balanced Dataset

Table B.3: Optimal Hyperparameters (Balanced Dataset)

Model	Optimal Hyperparameters
Random Forest	n_estimators=50 max_depth=10 min_samples_split=2 min_samples_leaf=1 max_features='sqrt' class_weight=None
Gradient Boosting	n_estimators=50 max_depth=3 learning_rate=0.1 subsample=0.8 max_features='sqrt'
LightGBM	n_estimators=100 max_depth=6 learning_rate=0.2

Continued on next page

Table B.3 – *Continued from previous page*

Model	Optimal Hyperparameters
	num_leaves=15 min_child_samples=20 class_weight='balanced' feature_fraction=0.8 bagging_fraction=0.8 bagging_freq=0 min_gain_to_split=0.0
XGBoost	n_estimators=200 max_depth=3 learning_rate=0.1 subsample=0.9 colsample_bytree=0.9 reg_alpha=0 reg_lambda=0
SVM (RBF)	C=1 gamma=0.1 kernel='rbf' class_weight=None probability=True
Logistic Regression	C=1 class_weight='balanced' max_iter=1000 penalty='l1' solver='liblinear'
Naive Bayes	var_smoothing=1e-09

Key Observations on Hyperparameter Differences

Class Weight Handling Models naturally adapt their class weighting strategies based on dataset balance. For the imbalanced dataset, Random Forest, SVM, and Logistic Regression utilize `class_weight='balanced'` to handle class imbalance, while this parameter is removed or set to `None` for the balanced dataset.

Model Complexity Adjustments Tree-based models show interesting complexity variations:

- **Random Forest:** Requires more trees for imbalanced data (200 vs 50 estimators)
- **Gradient Boosting:** Uses deeper trees for imbalanced data (depth=7 vs depth=3) but fewer estimators for balanced data (50 vs 200)
- **XGBoost:** Maintains consistent depth (3) but adjusts learning rate and regularization

Regularization Patterns

- **SVM:** Higher regularization strength ($C=10$) for imbalanced data vs lower ($C=1$) for balanced data
- **Logistic Regression:** Similar pattern with $C=10$ vs $C=1$, indicating need for stronger regularization with class imbalance
- **LightGBM:** Increases `min_child_samples` from 10 to 20 for balanced data, providing additional regularization

Consistency Across Datasets Naive Bayes shows remarkable consistency with identical hyperparameters (`var_smoothing=1e-09`) across both datasets, reflecting its probabilistic nature and robustness to class distribution changes.

Appendix C

Taint Analysis Sources and Sinks

C.1 Source APIs

This appendix contains the source and sink APIs for our static analysis in [Section 3](#)

C.1.1 Behavioral Source APIs

A complete list of behavioral sources can be found in our repository [\[7\]](#). These are all properties of classes derived from the `UIEvent` class in Chromium.

C.1.2 Fingerprinting Source APIs

Our system monitors a comprehensive set of JavaScript APIs commonly used for device fingerprinting. The complete list of fingerprinting APIs tracked by our detection system is available in our repository [\[7\]](#). This extensive collection includes APIs from various categories such as Navigator properties, Screen dimensions, Audio context information, HTML element measurements, WebGL parameters, and device-specific interfaces that can be used to uniquely identify or profile user devices. The APIs span multiple fingerprinting vectors including browser fingerprinting, canvas fingerprinting, audio fingerprinting, and hardware profiling techniques commonly employed by tracking scripts.

C.1.3 Sink APIs

Table C.1: Known data sinks monitored for behavioral biometric data exfiltration

Sink Category	API Endpoints
Local Storage	Window.sessionStorage, Window.localStorage, Window.openDatabase, Window.indexedDB
Database Operations	IDBObjectStore.put, IDBObjectStore.add
Network Communication	XMLHttpRequest.send, WebSocket.send, Navigator.sendBeacon, RTCDataChannel.send
Message Passing	MessagePort.postMessage, ServiceWorker.postMessage, Window.postMessage, Client.postMessage, DedicatedWorkerGlobalScope.postMessage, Worker.postMessage
DOM Manipulation	Document.cookie, HTMLInputElement.value, Node.textContent
Resource Loading	HTMLScriptElement.setAttribute, HTMLScriptElement.src, HTMLImageElement.src