Finding Behavioural Biometrics Scripts on the Web Using Dynamic Taint Analysis

by

Alexandru Bara

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

© Alexandru Bara 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

In an era of escalating cyber threats, behavioural biometrics have emerged as a transformative security mechanism, leveraging user interaction patterns like keystrokes and mouse movements for continuous authentication on the web. However, detecting these scripts at scale remains challenging due to obfuscation, dynamic execution, and overlap with analytics tools. This thesis addresses these challenges through three interconnected contributions: (1) enhancing FoxHound, a dynamic taint analysis tool, to achieve 97% effectiveness in tracking behavioural biometric data flows; (2) developing the first open-source checkout crawler to navigate e-commerce workflows with upwards of 78% accuracy; and (3) creating a machine learning classifier to distinguish behavioural biometric scripts from other tracking scripts. Large-scale analyses reveal that behavioural biometric scripts are deployed on 0.3% of top websites, with significantly higher adoption on sensitive pages (4.55% of banking logins). The work concludes with ethical recommendations to balance security benefits with privacy risks, advocating for transparency, deobfuscation, and regulatory oversight.

Acknowledgements

There are many individuals I would like to thank for their support throughout this journey. First and foremost, I am deeply grateful to my supervisor, Dr. Urs Hengartner, for his invaluable guidance and assistance during my thesis work.

I also wish to express my heartfelt appreciation to my family and Brianna Koti, whose unwavering support and belief in me have been a constant source of strength.

Special thanks to Aswad Tariq for his significant contributions along the way. I would not have been able to complete my master's degree if it weren't for the persistent encouragement of Jason Gerard, Armand Khairabadi, Daniel Santana, Sina Kamali, Haseeb Faheem, Karim Kati, Andre Ibrahim, and many others; I am deeply grateful for their support in pushing me to become better.

Dedication

This is dedicated to Brianna Koti.

Table of Contents

Author's Declaration ii

Abstract iii

Acknowledgements iv

Dedication v

List of Figures xii

List of Tables xiii

1 Introduction 1

1.1 The Rise of Behavioural Biometrics 1

1.2 Challenges in Detecting Behavioural Biometrics 2

1.3 Research Questions 2

1.4 Methodology 3

1.4.1 Dynamic Taint Analysis with FoxHound 3

1.4.2 Checkout Crawler 3

1.5 Contributions 4

1.6 Implications and Future Work 5

1.7 Thesis Structure 5

2 Background **7**
2.1 Dynamic Taint Analysis . . . 7
2.2 Web Crawlers . . . 10
2.2.1 Crawling Architectures . . . 10
2.2.2 Specialized Crawlers . . . 12
2.2.3 Summary . . . 14
2.3 Fingerprinting and Behavioral Biometrics Scripts on the Web . . . 14
2.3.1 Browser Fingerprinting . . . 14
2.3.2 Behavioral Biometrics Scripts . . . 16

3 Dynamic Taint Analysis **18**
3.1 Introduction . . . 18
3.2 Chapter Overview . . . 19
3.3 Dynamic Taint Analysis Tool . . . 19
3.3.1 Reason For Choosing FoxHound . . . 20
3.3.2 How FoxHound Works . . . 20
3.4 Modifications to FoxHound . . . 22
3.4.1 Methodology . . . 23
3.4.2 Neuro-Id . . . 26
3.4.3 Iovation . . . 26
3.4.4 BehavioSec . . . 28
3.4.5 BioCatch . . . 29
3.4.6 Accertify . . . 30
3.4.7 CallSign . . . 31
3.4.8 TypingDNA . . . 32
3.4.9 Transmit Security . . . 33
3.4.10 Forter . . . 34
3.4.11 Sardine.ai, f5, Human . . . 35

3.4.12 Darwinium, Feedzai, DataVisor, BureauID . . . 35
3.4.13 Modified FoxHound . . . 36
3.5 Discussion . . . 38
3.5.1 Effectiveness Improvements . . . 38
3.5.2 Observations . . . 38
3.5.3 Limitations . . . 40
3.6 Conclusion . . . 41

4 Checkout Crawler **42**
4.1 Introduction . . . 42
4.2 E-Commerce Website Flow . . . 43
4.2.1 Main Page . . . 44
4.2.2 Collection page . . . 44
4.2.3 Product Page . . . 44
4.2.4 Cart Page . . . 45
4.2.5 Checkout Page . . . 47
4.3 Crawler Design . . . 47
4.3.1 Infrastructure . . . 48
4.3.2 High Level Design . . . 48
4.3.3 Crawling Modules . . . 48
4.3.4 Page Gathering . . . 50
4.3.5 Size Selection Component . . . 51
4.3.6 Add to Cart Component . . . 51
4.3.7 Go to Cart Component . . . 51
4.3.8 Go to Checkout Component . . . 52
4.4 Evaluation . . . 52
4.4.1 Datasets . . . 52
4.4.2 Experiment . . . 53

4.4.3 Results . . . 54
4.5 Discussion . . . 54
4.5.1 Top 1000 Shopify . . . 54
4.5.2 Top 118 Random . . . 57
4.6 Future Work . . . 58
4.7 Conclusion . . . 59

5 Machine Learning Pipeline **60**
5.1 Introduction . . . 60
5.1.1 Script Variants in Web Analytics . . . 60
5.1.2 The Role of Machine Learning in Script Classification . . . 61
5.2 Building Ground Truth . . . 62
5.2.1 Simulated User Module . . . 62
5.2.2 Storage . . . 63
5.2.3 List of Websites . . . 63
5.2.4 Ground Truth . . . 64
5.3 Feature Extraction . . . 65
5.3.1 Feature Engineering . . . 65
5.3.2 Feature Overview . . . 66
5.4 Evaluation . . . 67
5.4.1 Random Forest . . . 68
5.4.2 Support Vector Machines (SVM) . . . 68
5.4.3 K-Nearest Neighbors (KNN) . . . 68
5.4.4 XGBoost . . . 69
5.4.5 Neural Networks . . . 69
5.5 Results and Discussion . . . 69
5.5.1 Results . . . 69
5.5.2 Model Trade-Offs and Justifications . . . 70
5.5.3 Threats to Validity . . . 72
5.6 Conclusion . . . 72

6 Results **74**
6.1 Introduction . . . 74
6.2 BBAT . . . 74
6.3 Analysis . . . 75
6.4 Results . . . 77
6.4.1 Study 1 . . . 77
6.4.2 Study 2 . . . 78
6.4.3 Study 3 . . . 78
6.4.4 Behavioural Biometrics Scripts . . . 80
6.5 Discussion . . . 80
6.5.1 Research Questions . . . 81
6.5.2 Extra Findings . . . 84
6.6 Conclusion . . . 87

7 Conclusion **88**
7.1 Contributions . . . 88
7.2 Key Findings . . . 89
7.3 Recommendations for Ethical Deployment . . . 89
7.4 Future Directions . . . 91

References **92**

APPENDICES **98**

A Scripts **99**
A.1 Web Analytics Scripts . . . 99
A.2 Behavioural Biometric Scripts . . . 99

B Dynamic Taint Analysis **104**
B.1 Behavioural Biometrics Sources and Sinks . . . 104

C E-Commerce **107**

C.1 E-Commerce Websites 107

List of Figures

3.1 High level overview of Firefox architecture [24] 21

3.2 Typical flows for behavioural biometrics scripts 25

3.3 Obfuscated script from the company Human. 39

4.1 E-Commerce checkout flow 43

4.2 Main pages examples. From left to right, Elgato's website has promotional content. Ring's website shows different collections. Keen's website has products that can be bought directly. 44

4.3 Ring's alarms collection page with filtering abilities on the left. 45

4.4 A product page from Keen's with a product description, the ability to select size and to add the product to the cart. 46

4.5 Keen's has both a cart page and a cart banner. Both have the checkout keyword. 46

4.6 Keen's checkout page. 47

4.7 Crawling Modules. From left to right: Simple module, cart module, size module and cart and size module 49

List of Tables

2.1 Dynamic Taint Analysis Tools 10
2.2 Comparative Analysis of Crawling Infrastructures 12

3.1 Taint flows caught by FoxHound without our changes 24
3.2 Taint flows caught by FoxHound with our changes 37

4.1 Checkout Crawler Results 54

5.1 Extracted Features from Taint Reports 67
5.2 Evaluation Metrics for Machine Learning Models 70

6.1 Common crawl messages and their counts for the top 20k websites 77
6.2 Common crawl messages and their counts for the 500 bank websites main page 78
6.3 Common crawl messages and their counts for the 1355 bank websites login page 79
6.4 Common crawl messages and their counts for the 1489 random websites login page 79
6.5 Common crawl messages and their counts for the top 1000 Shopify websites 80
6.6 Common crawl messages and their counts for 118 e-commerce websites . . 80
6.7 E-commerce and Shopify checkout page status 81
6.8 Behavioural Biometrics scripts for all studies 82
6.9 Behavioural Biometrics scripts for all studies 83

A.1 Websites containing various analytic scripts. 99
A.2 Behavioural biometrics companies and websites that contain their scripts . 103
B.1 List of sinks and sources used in taint analysis to find Behavioural Biometrics scripts 106
C.1 List of 118 curated e-commerce websites 108

Chapter 1

Introduction

1.1 The Rise of Behavioural Biometrics

In an era of escalating cyber threats, traditional authentication mechanisms like passwords and two-factor authentication (2FA) increasingly fall short of addressing sophisticated attacks such as credential stuffing, session hijacking, and social engineering. Behavioural biometrics have emerged as a transformative solution, leveraging unique user interaction patterns like keystroke dynamics, mouse movements, touchscreen gestures, and navigation behaviours to authenticate users transparently and continuously. Unlike static credentials, behavioural biometrics operate in the background, analyzing real-time interactions with a website to detect anomalies that may signal fraud. For instance, a banking platform might flag a transaction if the user's typing rhythm deviates from their historical profile, even if their password is correct.

This technology is particularly critical for securing high-risk workflows, such as financial transactions, healthcare portals, and enterprise systems. However, its adoption raises significant questions: How pervasive are these behavioural biometrics scripts? Where are they deployed? Do they prioritize sensitive pages like logins or checkouts? Answering these questions is essential for understanding modern web security practices and their privacy implications.

1.2 Challenges in Detecting Behavioural Biometrics

Despite their growing popularity, behavioural biometrics scripts remain notoriously difficult to detect. Three key challenges hinder their analysis:

- Dynamic Execution: These scripts often operate dynamically, collecting data only during specific user interactions (e.g., form submissions and mouse movement).
- Obfuscation: Vendors heavily obfuscate code to prevent reverse engineering, masking data flows even from advanced static analysis tools.
- Overlap with Analytics: Behavioural biometrics scripts share data collection patterns with session replay and heatmap tools, complicating classification.

Existing studies have focused on fingerprinting and tracking scripts but there has not been any approach that looks for behavioural biometrics scripts. A systematic, scalable approach to detect and analyze these scripts across the broader web is absent and this thesis seeks to fill this gap as well as provide a ground truth for behavioural biometrics scripts.

1.3 Research Questions

This work addresses six research questions (RQs) to map the deployment and operational logic of behavioural biometrics scripts:

- Prevalence: How widespread are behavioural biometrics scripts on the web? (**RQ1**)
- Sensitive Pages: Are these scripts disproportionately deployed on pages handling sensitive data (e.g., logins and checkout pages)? (**RQ2**)
- Login vs. Main Pages: Are login pages more likely to host behavioural biometrics scripts than main pages? (**RQ3**)
- Banking Logins: Do banking login pages employ these scripts more frequently than non-banking logins? (**RQ4**)
- Checkout Pages: Is checkout page scripting more prevalent than on other e-commerce pages? (**RQ5**)
- Shopify Support: Does Shopify, the leading e-commerce platform, enable behavioural biometrics scripting? (**RQ6**)

1.4 Methodology

To answer these questions, we develop three interconnected components:

1.4.1 Dynamic Taint Analysis with FoxHound

We enhance FoxHound [32], an open-source dynamic taint analysis tool, to track fine-grained data flows from user interactions (sources) to network exfiltration points (sinks). Key modifications include:

- Boolean Tainting: Extending taint propagation to previously unsupported data types.
- Inter-Process Communication: Ensuring taint metadata survives serialization between main scripts and web workers.
- Obfuscation Resilience: Ensuring proper taint propagation in obfuscated scripts, which tend to use JavaScript functionalities not used in normal scripts.

These enhancements boost FoxHound's detection effectiveness from 20% to 97% across 15 real-world behavioural biometrics scripts.

1.4.2 Checkout Crawler

To analyze e-commerce workflows, we design a novel checkout crawler that navigates from a website's homepage to its checkout page with accuracy as high as 78% on Shopify stores. The crawler:

- Simulates user actions (e.g., product selection, cart additions).
- Bypasses bot detection using stealth plugins and adaptive heuristics.
- Has four modules to cover a wide array of e-commerce websites.

We also noticed that there was not a good e-commerce website dataset and we created the first ever e-commerce dataset, which consists of 118 websites that have been tested and confirmed to be e-commerce websites.

To distinguish behavioural biometrics scripts from similar tracking mechanisms (e.g., session replay tools), we developed a Random Forests-based classifier trained on a dataset of 126 manually labelled scripts. Each script is characterized by a set of features extracted from dynamic taint analysis, which capture the complexity and behaviour of data flows.

Key features include:

- **Taint Flow Complexity:** Metrics such as the total node count in the taint propagation graph and the diversity of operations encountered. These features quantify the intricate data flows that are characteristic of behavioural biometrics scripts.
- **Source-Sink Frequency:** The frequency with which various event sources (e.g., keyboard, mouse, and touch events) reach designated sinks (e.g., network endpoints, sockets, and HTTP requests). This metric provides a measure of how actively a script captures and transmits user interaction data.

Our experiments involved testing several machine learning classifiers, and the best-performing model is a random forest that achieved an overall accuracy of 96%. However, signs of overtraining were observed, suggesting that the model's generalization ability could be improved.

1.5 Contributions

This thesis makes four key contributions:

- **Tooling**: A significantly enhanced FoxHound browser, the first open-source checkout crawler and a machine learning model that is tailored for behavioural biometrics analysis. We combined these three components to create BBAT [7], our open-source behavioural biometrics analysis tool.
- **Dataset**: A set of hundreds of tracking scripts manually classified as either behavioural biometrics or not.
- **Empirical Insights**: We show that behavioural biometrics are deployed on 0.3% of top websites, with banking logins and Shopify checkouts showing a significantly higher adoption than main pages.
- **Ethical Framework**: Recommendations for balancing security benefits with privacy risks, particularly regarding data collection and obfuscated scripts.

1.6 Implications and Future Work

Our findings highlight the dual-edged nature of behavioural biometrics: While they enhance security, their invasive tracking risks breaking user trust. We advocate for:

- Transparency: Mandating that organizations must explicitly notify users when such data is being gathered, detailing the nature of the data, the methods of collection, the intended use, and any third parties with whom the data may be shared.
- Regulation: Extending GDPR/CCPA to cover biometrics data flows. Extending these regulations to cover biometrics data flows involves ensuring that any collection, storage, processing, or sharing of biometrics data adheres to the principles of lawfulness, fairness, and transparency.
- Unobfuscation: Unobfuscated behavioural biometrics scripts enhances transparency, allowing users to understand the specific data being collected and how it is utilized. Given that the core functionalities of these scripts typically reside on backend servers, obfuscating the client-side code offers minimal security benefits.

Future work could explore longitudinal studies to track adoption trends. A larger scale study can also be conducted to get a better idea on the impact of behavioural biometrics scripts. Another avenue to explore is the ground truth. We currently are rellying on a very limited ground truth to train our machine learning model and a more comprehensive ground truth will help in increasing the accuracy of our study.

1.7 Thesis Structure

- Chapter 2 gives background and related work on behavioural biometrics scripts
- Chapter 3 details FoxHound's enhancements and evaluation.
- Chapter 4 presents the checkout crawler's design and performance.
- Chapter 5 discusses the machine learning pipeline and classification results.
- Chapter 6 discusses findings from large-scale web analyses.
- Chapter 7 concludes with broader implications and policy recommendations.

This work bridges technical innovation with societal impact, offering a baseline for detecting and ethically governing behavioural biometrics technologies in an increasingly adversarial digital landscape.

Chapter 2

Background

This section will give all the relevant context to follow the work done in this thesis.

2.1 Dynamic Taint Analysis

Dynamic taint analysis enables runtime analysis of scripts using tainted data. Data originating from sources is marked as tainted, and when this data reaches a sink, a taint report is generated. This technique provides valuable insights into how data flows through applications, identifying potential vulnerabilities and misuse.

Various dynamic taint analysis tools exist, each tailored to different use cases and capabilities. To narrow our scope, we focus on tools designed for JavaScript that can operate within a browser environment. Key criteria for selecting a suitable tool include:

- **Taint Tracking Granularity**: The level at which taint is propagated, with finer granularity (e.g., character-level tainting for strings) offering more precision compared to coarser granularity (e.g., object-level tainting).
- **Taint History**: The ability to generate detailed reports of the data's journey through the code (taint flow), from source to sink.
- **Customizable**: Flexibility in defining custom sources and sinks.
- **Reproducible**: Open source and ease of setup across different systems.

- **Actively Maintained**: The tool is still being worked on, bugs are actively fixed and help is provided with issues regarding the tool.
- **Supported Types**: Compatibility with key JavaScript types, including strings, numbers, objects, and booleans.

FoxHound

FoxHound [32] was created to detect and validate client-side JavaScript sanitizer functions to detect potential cross-site scripting attacks. FoxHound has proven to be a useful tool for dynamic taint analysis and has been used for cookie banner detection [33] and browser fingerprinting detection [12]. FoxHound has fine granularity since it provides taint tracking at the character level. It also provides the history of taint flows with the ability to have multiple sources reach a sink. The sources are easy to modify in the webidl files and the tool is open source. Foxhound provides taint support for strings, numbers and objects. Booleans are not taint-aware.

PanoptiChrome

PanoptiChrome is a new dynamic taint analysis tool built for the latest version of Chrome and has been used to find browser fingerprinting APIs [30]. PanoptiChrome allows for taint tracking of individual JavaScript objects, making it coarse granularity. PanoptiChrome provides a list of sources that get to a sink but does not provide the taint flow. It allows the user to specify which sources to track and supports all JavaScript types. We tried to build the system but we were not able to. We emailed the authors and we got an answer saying that a generic installer does not currently work. As a result, PanoptiChrome is not reproducible and cannot function across different systems.

Mystique

Mystique modifies the V8 engine from Chrome to provide taint tracking capabilities, which was used to find leakage in browser extensions [14]. Mystique allows for tainting at the object level, which makes it coarse-grained. Mystique has a taint table to keep track of which object is tainted and the values in the taint tables are updated whenever tainted objects are used. This architecture cannot allow for character-level tainting. Mystique does offer taint flow history when reports are created but it does not offer customizable sources.

We were able to build and reproduce the Mystique system and all types are supported for taint propagation.

25 Million Flows Later

Lekies et al. [36] created a dynamic taint tracking system to find DOM-based XSS vulnerabilities. They modified the Chrome engine and changed the string representation to allow for byte-wise tainting giving this approach fine granularity. The tool only reports which sources went to which sinks and there is no history kept. The sources and sinks, which are set with no clear path to customization, limit the tool's flexibility. The only types that are supported are strings and objects, there is no primitive type tainting. Lastly, there is no open-source implementation of the tool, which makes it non-reproducible.

Platform-Independent Dynamic Taint Analysis

Karim et al. [31] created a platform-independent dynamic taint analysis tool. The main concept relies on source code instrumentation given by a taint specification that contains sources and sinks. An instrumentation framework is used, which creates the taint propagation. That is then encoded as instructions for an abstract machine. This technique allows all types to be tainted and it does so at the object level making it coarse-grained. The taint reports also do not contain information about the history. This tool is not open source and we cannot reproduce it.

LeakTracker

LeakTracker [52] is a tool that enhances the Firefox JavaScript engine to support tainting. The main idea behind LeakTracker is to be more efficient than other taint tools, which is done by only tainting only on suspicious variables. The tainting is done at the variable level making it coarse grained. This tool is a prototype and it only offers four sinks and six sources, which cannot be customized. It also does not report the taint history and it is not open source. There is no mention of what types are supported by this tool. There is no mention of primitive types tainting, which makes the supported types for tainting strings and objects.

Tool	Granularity	History	Customizable	Reproducible	Supported Types
FoxHound [32]	Fine	Yes	Yes	Yes	Strings, Numbers, Objects
PanoptiChrome [30]	Coarse	No	Yes	No	All
Mystique [14]	Coarse	Yes	No	Yes	All
Lekies et al. [36]	Fine	No	No	No	Strings, Objects
Karim et al. [31]	Coarse	No	Yes	No	All
LeakTracker [52]	Coarse	No	No	No	Strings, Objects

Table 2.1: Dynamic Taint Analysis Tools

Tools Summary

Table 2.1 summarizes the tools and their attributes, highlighting their benefits and limitations. While several tools offer unique capabilities, FoxHound's fine-grained granularity, taint flow history, customizability, and active maintenance make it the ideal choice for this thesis.

2.2 Web Crawlers

2.2.1 Crawling Architectures

Web crawlers, often referred to as spiders, are automated programs designed to systematically explore and index web content. They serve as the backbone for search engines and data extraction systems by continuously discovering new and updated information on the web. A typical crawler architecture comprises three core components:

- **URL Frontier:** This component manages the queue of web pages to visit. It employs various strategies—including priority scheduling and politeness policies, such as rate limiting—to avoid overwhelming target servers [47].
- **Downloader:** Responsible for fetching web content via HTTP/HTTPS requests, modern downloaders leverage tools like `Playwright` [40] and `Selenium` [15] to emulate

full browser environments. This capability is critical for rendering JavaScript-heavy single-page applications (SPAs) and dynamic content.

- **Parser:** Once a page is retrieved, the parser extracts structured data and hyperlinks using techniques that range from traditional DOM tree analysis to advanced machine learning models for semantic extraction [16].

Modern crawling faces several challenges. Dynamic content—such as infinite scrolling or AJAX-driven interfaces—necessitates that crawlers simulate human interactions like clicking and scrolling. Additionally, advanced bot mitigation measures, including fingerprinting, CAPTCHAs, and behavioural analysis, have been widely deployed. Ethical considerations, such as strict adherence to `robots.txt` directives and compliance with GDPR when handling personal data, further complicate the crawling process [23].

There exists a diverse array of crawling infrastructures developed to facilitate research. An ideal crawling system should allow for flexible browser selection, ease of deployment, straightforward extendibility, robust fault tolerance (via retry mechanisms), and checkpointing to prevent loss of crawl progress. Table 2.2 provides a comparative overview of several notable tools and highlights their respective strengths and limitations.

FP-Tracer

FP-Tracer [12] is a crawling tool tightly integrated with FoxHound. It offers robust fault tolerance through built-in retry mechanisms and checkpointing, ensuring that crawl progress is maintained despite interruptions. However, its limited ease of use and poor extendibility may restrict its adaptability for broader research applications.

Tracker Radar Collector

Developed by DuckDuckGo [18], Tracker Radar Collector is designed with simplicity and adaptability in mind. Its user-friendly interface makes it accessible to a wide range of users. Nonetheless, it lacks native retry mechanisms and checkpointing. It also does not integrate with FoxHound, which can be a limitation for certain advanced analyses.

OpenWPM

OpenWPM [21] is a widely adopted framework for automated web measurements. Its simplicity and ease of deployment make it popular among researchers. However, its extendibility is limited, and it does not support built-in retry mechanisms or checkpointing.

Tool	Ease of Use	Extendible	Retry Mechanism	Checkpointing	Supports FoxHound
FP-Tracer [12]	No	No	Yes	Yes	Yes
Tracker Radar Collector [18]	Yes	Yes	No	No	No
OpenWPM [21]	Yes	No	No	No	No
Klein et al. [33]	Yes	Yes	Yes	Yes	Yes

Table 2.2: Comparative Analysis of Crawling Infrastructures

Furthermore, it lacks integration with FoxHound, which may be essential for specific use cases.

Klein et al.

The crawling infrastructure developed by Klein et al. [33] provides a comprehensive and modular framework. It excels in ease of use, extendibility, and robust fault tolerance through integrated retry mechanisms and checkpointing. Its seamless support for FoxHound integration makes it an ideal choice for complex, large-scale crawling tasks, and for our research objectives.

Considering the essential requirements for flexible browser selection, ease of operation, extendibility, robust fault tolerance via retry mechanisms and checkpointing, and full support for FoxHound, Klein et al.'s tool [33] stands out as the most suitable choice. Its comprehensive feature set and modular design render it ideally suited to meet our research objectives.

2.2.2 Specialized Crawlers

There are many types of crawlers, each designed to address specific tasks. Our study focuses on sensitive pages that contain private information such as passwords and credit card numbers. Accordingly, we differentiate between two specialized crawler types: login crawlers and checkout crawlers.

Login Crawlers

Login crawlers specialize in automating authentication workflows to analyze security mechanisms and access restricted content. Numerous login crawlers exist; here we highlight some of the latest open-source approaches:

- **SSO-Monitor** [29]: This work provides a comprehensive survey of Single Sign-On (SSO) systems, emphasizing current security challenges and future research directions. Its accompanying crawler identifies and analyzes SSO login pages and authentication flows, facilitating vulnerability assessment and security measurement.
- **Innocenti et al.** [27]: Focusing on OAuth 2.0, this study scrutinizes the security of redirect URI validation. The proposed crawler systematically detects weaknesses in OAuth flows, identifying misconfigurations that could lead to phishing, session hijacking, or other security breaches.
- **The Double-Edged Sword** [45]: This research examines how authentication pages can inadvertently leak identifying information via fingerprinting. The crawler automatically locates and classifies authentication pages based on layout, interactive behaviour, and embedded fingerprinting elements, revealing how design choices may compromise user privacy.

While all three crawlers are open source, they primarily rely on heuristic and regular expression-based methods. When multiple login pages are present, these tools typically return a single candidate rather than supporting multiple outputs. To address this limitation, Tariq [49] extended the SSO-Monitor crawler by integrating large language models for improved crawl and classification accuracy, as well as support for multiple login candidates. Consequently, we will adopt Tariq's enhanced crawler [49] for our login page detection tasks.

Checkout Crawlers

To the best of our knowledge, there is currently no checkout crawler that can navigate any e-commerce website to its checkout page. Our work is the first to develop such a crawler. Our novel implementation, which simulates a three-stage workflow—navigating to the product page, adding the item to the cart, and proceeding to the checkout page—achieved an accuracy of 78% on Shopify stores. Detailed implementation aspects are discussed in later chapters.

2.2.3 Summary

Specialized crawlers enable a granular analysis of authentication and payment workflows. While general-purpose crawlers offer broad coverage, tools such as login and checkout crawlers provide in-depth insights through domain-specific strategies. For our study, we will employ Tariq's login crawler and introduce the first-ever checkout crawler, advancing the state-of-the-art in automated web analysis.

2.3 Fingerprinting and Behavioral Biometrics Scripts on the Web

JavaScript is a versatile, high-level programming language that forms the cornerstone of modern web development. Initially designed to add interactivity to static HTML pages, JavaScript has evolved into a powerful tool that drives dynamic, responsive, and feature-rich web applications. This section examines the role and usage of JavaScript scripts on the web, with a particular focus on their implications for security and privacy.

2.3.1 Browser Fingerprinting

Browser fingerprinting scripts enable the tracking of users across websites by exploiting subtle differences in data exposed through browser APIs. These scripts systematically query a range of APIs that return values indicative of the user's unique browser environment. For example, a call to the `screen.height` API yields the height of the user's display. While such a value might be common among many users, when combined with additional attributes—such as the browser version, installed plugins, font lists, and other configuration details—the resulting composite profile becomes highly distinctive.

Story of Browser Fingerprinting

The concept of leveraging web APIs to de-anonymize users was pioneered by Mayer in 2009 [38]. In his seminal work, Mayer examined several browser objects—such as `navigator`, `screen`, `plugins`, and `mimeTypes`—and demonstrated that 1,278 out of 1,328 clients (96.23%) could be uniquely identified. This groundbreaking study laid the foundation for the field of browser fingerprinting.

Building on Mayer's initial findings, Eckersley conducted a large-scale study in 2010 [20], titled *How Unique Is Your Web Browser?*. By capturing 470,161 fingerprints, Eckersley showed that 94.2% of them were unique, effectively coining the term "browser fingerprinting" and popularizing the idea of using these techniques as an alternative to cookie-based tracking.

Subsequent research has expanded the array of fingerprinting vectors. Mowery et al. [41] demonstrated that the `canvas` and `WebGL` APIs could be used as effective fingerprinting attributes, while Englehardt et al. [21] identified the `AudioContext` API as another potent vector for fingerprinting. Additionally, Sjösten et al. [48] revealed that the detection of browser extensions could further contribute to user identification. Other studies have also highlighted the potential of CSS properties [53] and battery status [43] as fingerprinting tools.

More recent analyses have shown that the scope of fingerprinting has expanded significantly, with studies reporting the use of upwards of 262 different fingerprinting APIs [6]. These advancements underscore both the increasing sophistication of browser fingerprinting techniques and the growing challenges they pose for user privacy.

Prevalence of Browser Fingerprinting Scripts

Initially, browser fingerprinting scripts were relatively uncommon on the web. In 2013, Nikiforakis et al. [42] demonstrated that only 0.4% of 10,000 websites employed fingerprinting scripts. The following year, Acar et al. [1] found that 5.5% of 100,000 websites used some form of fingerprinting techniques. Over time, however, the adoption of these scripts has increased substantially. In 2021, Iqbal et al. [28] observed that 30.6% of the top 1,000 websites deployed fingerprinting, although this percentage dropped to 10.18% when considering the top 100,000 websites—indicating that more popular websites are particularly prone to using these techniques. More recently, in 2024, Boussaha [12] reported that, under a relaxed definition of fingerprinting, approximately 75% of websites employ such techniques. These findings underscore a rapid increase in both the prevalence and sophistication of browser fingerprinting over time.

Security and Privacy Tradeoff

While browser fingerprinting is often criticized for its potential to compromise user anonymity, it also offers significant security benefits. Laperdrix et al. [34] discuss how these techniques can be effectively employed for bot detection, vulnerability patching, and even enhancing

authentication processes. In support of this view, Antonin et al. [19] explored the integration of browser fingerprinting into multi-factor authentication, reporting that a dozen security-oriented companies actively collect fingerprint data during sign-up, sign-in, and payment procedures. Moreover, Andriamilanto et al. developed BrFAST [5], a tool that empowers organizations to select specific fingerprinting attributes to optimize the balance between security and usability in their authentication systems. More recently, Senol et al. [45] demonstrated that login pages are more likely to incorporate fingerprinting scripts compared to standard pages, highlighting the strategic use of these scripts in improving security measures. Collectively, these studies illustrate that, when implemented responsibly, browser fingerprinting can serve as a valuable component in comprehensive security frameworks.

2.3.2 Behavioral Biometrics Scripts

In this work, we define behavioural biometrics scripts as scripts that leverage the way users interact with websites to identify them. This interaction encompasses keystroke patterns, mouse movements, and touch dynamics. To the best of our knowledge, no previous research has specifically examined these scripts or their prevalence on the web.

Mechanisms of Behavioral Biometrics Scripts

Understanding the operation of behavioural biometrics scripts requires an examination of mouse, keystroke, and swipe dynamics, along with the various authentication systems developed around them.

Keystroke dynamics were first introduced by Gaines et al. [22] in 1980, who analyzed the timing of individual keystrokes to differentiate users from a small sample. Subsequent studies have explored additional features such as the duration a key is held down and the time interval from the initiation to the full depression of a key. More recent approaches have incorporated machine learning models to enhance user differentiation [50]. For example, Samura et al. [44] demonstrated that using a support vector machine (SVM) allowed them to distinguish among 112 users with 100% accuracy, showcasing the potential of this authentication method.

Mouse dynamics emerged later than keystroke dynamics. In 2007, Ahmed et al. [3] were the first to propose a system utilizing artificial neural networks to distinguish users, achieving a false acceptance rate (FAR) of 2.4649% and a false rejection rate (FRR) of 2.4614%. Subsequent research examined additional features such as movement offset and

curvature speed to build more robust models [51]. Notably, Shen et al. [46] managed to reduce the FAR to 1.18% and the FRR to 1.96%. Similarly, swipe patterns, which are analogous to mouse dynamics but designed for touchscreen devices, have been employed in authentication systems [26].

Behavioural biometrics scripts operate by first capturing mouse, keyboard, and touch events via active event handlers. This data is then aggregated and transmitted from the webpage to backend servers, where models employing keystroke, mouse, and swipe dynamics are used to validate the user.

The Necessity of Behavioral Biometrics Scripts

While existing research has shown that browser fingerprinting can enhance security on sensitive pages, such as login screens, behavioural biometrics scripts provide an additional layer of protection. To illustrate their importance, consider an attack on browser fingerprinting.

Lin et al. [37] demonstrated a phishing attack where a malicious website harvested a user's email, password, and browser fingerprint. By altering their fingerprint to mimic the victim's, the attackers bypassed the browser fingerprinting security measures deployed on some login pages.

By incorporating behavioural biometrics scripts, such attacks can be mitigated, as an attacker would be unable to replicate the authentic behavioural patterns exhibited by the legitimate user during login.

Behavioural biometrics offer more than just enhanced authentication; they are also valuable for distinguishing between real users and bots on the web. Additionally, they play a crucial role in preventing various types of fraud, including click fraud and payment fraud. As a relatively new field, behavioural biometrics hold immense potential for advancing security measures, with promising opportunities yet to be fully explored.

Chapter 3

Dynamic Taint Analysis

This chapter will discuss what dynamic taint analysis is and how it can help us find behavioural biometrics scripts.

3.1 Introduction

To understand the need for dynamic taint analysis, we first need to understand what taint analysis is. DeepSource defines it as "a process used in information security to identify the flow of user input through a system to understand the security implications of system design" [17]. When using taint analysis, we usually have taint sources, which are the data that we want to track through an application. The taint flow is the path in our app through which our tainted data goes through. We also have taint sinks, which are the points in our application where we want to stop following the taint flow.

One of the key characteristics of behavioural biometrics scripts is that they use user behaviour to decide whether there has been an account takeover. On a computer, the two main features that behavioural biometrics track are mouse movements and keyboard interactions. A behavioural biometrics script will usually have event listeners for keyboard events and mouse events. It will then take the properties of the events like the mouse position or the key press and it will send the data to its servers to decide whether the user is who they say they are. Taint analysis can be used in this situation to find which scripts are behavioural biometrics scripts. To do this, we can mark mouse and keyboard events as sources. We can then proceed to mark the known exit points of a script as taint sinks. The taint flows produced by various scripts can then be used for script classification purposes.

Looking at scripts can be done statically or dynamically. In the static way, we look at the script without executing it. We can mark taint sources and sinks and look at the flow by analyzing the code. This, however, does not take into consideration if the code gets executed. So, there may be a script on a webpage that is not active, and static analysis will capture it. Static analysis also fails when the code is heavily obfuscated. It becomes extremely difficult to mark sources and sinks. It also makes it difficult to follow the taint flow and gain insightful information. Dynamic analysis involves running the code and looking at the behaviour of the code when it is being executed. Dynamic analysis will ensure that only the executed code is taken into consideration since the paths not taken will not be executed. It will also not have any problem with obfuscated code since the code complexities will be resolved at runtime.

We believe that the best way to find behavioural biometrics scripts and companies that are using them is through dynamic taint analysis. Taint analysis is needed to see the flow of data that is relevant to us. The dynamic part is crucial to determine if the scripts are being used and if the data goes to the sinks. This chapter will discuss how we chose the dynamic taint analysis tool to use the modifications we made to it to fit our use case.

3.2 Chapter Overview

This chapter presents the enhancements made to FoxHound, a dynamic taint analysis tool, to address its limitations in tracking taint flows in behavioural biometrics scripts. We detail the modifications introduced, including support for boolean tainting, enhanced taint propagation, and improved inter-process communication handling. These changes increased FoxHound's effectiveness from 20% to 97%, enabling it to track taint flows in 14.5 out of 15 active scripts.

The chapter also discusses the challenges faced during the modifications, such as handling obfuscated scripts, managing memory overhead, and ensuring compatibility with JavaScript's dynamic nature. Finally, we evaluate the improved tool's performance, compare it with its original state, and classify its effectiveness across various real-world scripts.

3.3 Dynamic Taint Analysis Tool

In this section, we will discuss which dynamic taint analysis tool we picked for our study as well as give a high-level overview of how it works.

3.3.1 Reason For Choosing FoxHound

We selected FoxHound [32] as our dynamic taint analysis tool for several compelling reasons. One of FoxHound's standout features is its fine-grained tainting capability, allowing taint tracking at the character level for strings. This precision is critical for accurately identifying and analyzing taint flows in complex scenarios. Additionally, FoxHound supports taint tracking across various data types, including strings, numbers, and objects, making it versatile and well-suited for our needs.

Another significant advantage of FoxHound is its flexibility in customizing taint sources. This feature enabled us to tailor the tool to our specific use case, ensuring that all relevant data flows were captured. Furthermore, as a modification of Firefox, FoxHound is platform agnostic, inheriting Firefox's ability to run seamlessly on multiple operating systems. This cross-platform compatibility ensured ease of use and integration into our existing workflows.

FoxHound is actively maintained by a dedicated team that ensures the tool remains robust and up-to-date. Their assistance was invaluable in familiarizing us with FoxHound and supporting the implementation of necessary fixes and enhancements.

These features and the support we received made FoxHound the ideal choice for our dynamic taint analysis, enabling us to effectively track and analyze taint flows in behavioural biometrics scripts.

3.3.2 How FoxHound Works

FoxHound is a modified version of Firefox that incorporates taint tracking to monitor the flow of sensitive data throughout the browser. As shown in Figure 3.1, Firefox's architecture is composed of several key components.

The SpiderMonkey engine is Firefox's JavaScript engine, responsible for parsing, interpreting, and executing JavaScript code. On the other hand, Gecko acts as Firefox's layout and rendering engine, managing tasks such as HTML/CSS rendering, DOM manipulation, networking, and graphics rendering.

FoxHound introduces taint awareness into both SpiderMonkey and Gecko, enabling precise tracking of data as it moves through various browser subsystems. To illustrate how this taint awareness is implemented, we can examine the modifications made to the internal string representation in SpiderMonkey.

Strings in FoxHound have been enhanced with a pointer to taint ranges—data structures that record taint flows for specific character ranges within a string. These taint ranges

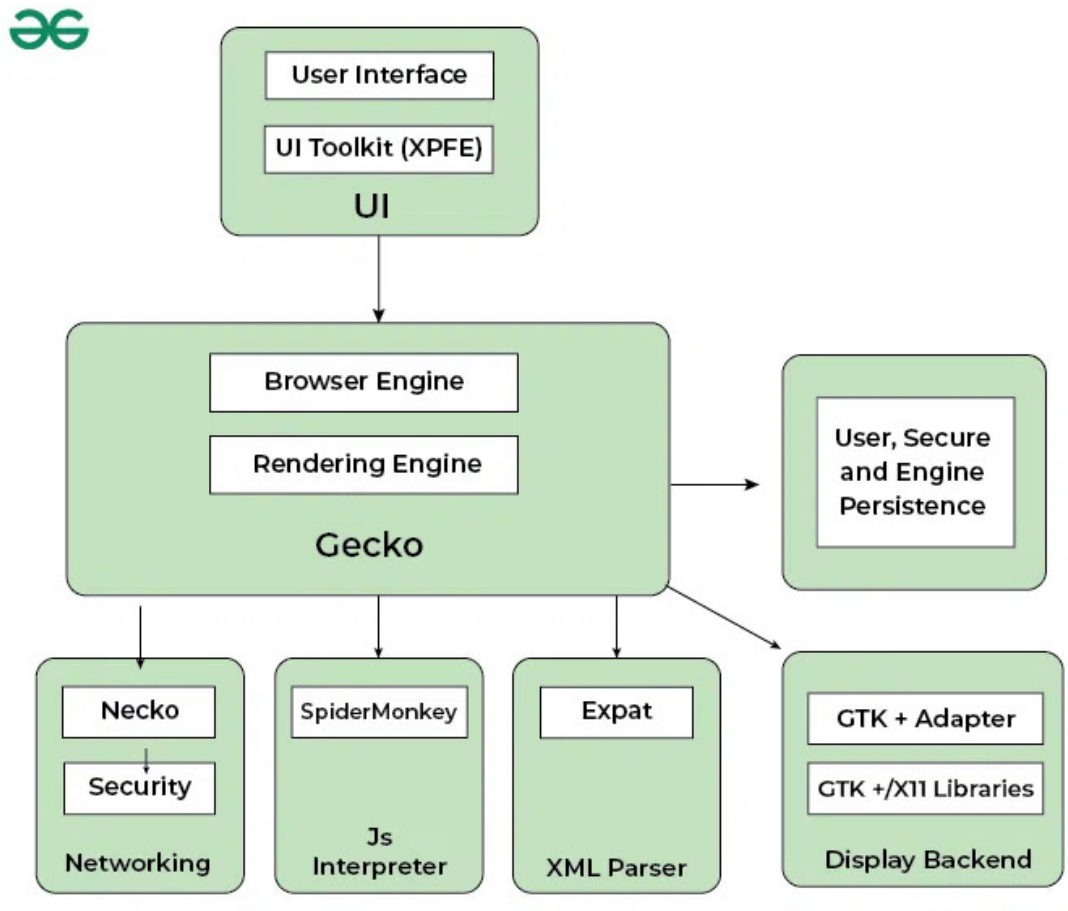


Figure 3.1: High level overview of Firefox architecture [24]

enable fine-grained taint tracking, allowing individual portions of a string to carry unique taint information. If the taint ranges pointer is NULL, it indicates that the string is not tainted. Conversely, a tainted string may contain one or more taint ranges, each associated with a distinct taint flow [32].

Each taint flow stores critical metadata, including the operations performed on the tainted data and the source responsible for introducing the taint. This design not only supports fine-grained tracking but also provides a comprehensive history of how tainted data has propagated and been manipulated within the browser. The same idea has also been applied to numbers to make them taint aware.

Additionally, FoxHound enhances the communication between these two engines to preserve taint metadata during inter-process interactions. This integration ensures that data originating from sources, such as user inputs, can be accurately tracked as it propagates through the browser, eventually reaching sinks like network requests.

3.4 Modifications to FoxHound

To enhance FoxHound's ability to track taint flows in behavioural biometrics scripts, we introduced several targeted modifications. These changes addressed specific limitations and significantly improved the tool's effectiveness:

- Boolean Tainting:
  - Added taint propagation for boolean values, which were previously unsupported.
  - Resolved issues with distinguishing between tainted boolean primitives and boolean objects without disrupting JavaScript behaviour.
- Math Library Enhancements:
  - Implemented taint propagation in the built-in Math library, ensuring that operations involving tainted numeric values maintain their taint metadata.
- Typed Arrays:
  - Marked typed arrays as sinks to handle preprocessing scenarios where behavioural biometrics data is staged before exfiltration.
- Inter-Process Communication:

  - Enabled taint tracking across web worker communication by modifying the serialization and deserialization processes in SpiderMonkey.

- Taint Propagation:
  - Found places where the taint was lost and ensured that it was properly propagated.

- Sink Detection:
  - Improved sink detection mechanisms to account for edge cases, such as hidden form elements accessed in unique ways by scripts like TypingDNA.

Many of these changes have already been integrated into FoxHound, while the remaining ones are currently awaiting approval. Below we will discuss our methodology for finding all the issues with FoxHound and how we were able to fix them in a more detailed manner.

3.4.1 Methodology

To ensure proper propagation of taint, we first need to test the base version of FoxHound, which does not include our modifications, against active scripts. The initial step involves identifying these active scripts. To accomplish this, we conducted joint work with Aswad Tariq aimed at discovering behavioural biometrics scripts and the websites that implement them.

We began by searching the web for companies that claim to offer behavioural biometrics solutions. Each company was analyzed to confirm that their definition of behavioural biometrics aligned with ours. Once verified, we attempted to locate their scripts by examining official documentation and identifying websites explicitly listed as customers. After obtaining the scripts, we extracted distinctive keywords, which were then used to search across a large dataset of websites and their corresponding scripts.

This allowed us to identify 19 behavioural biometrics companies, each associated with at least one website. The complete list of associated scripts is provided in Appendix A.

For our experiments, we selected one representative website per behavioural biometrics company to test with FoxHound. Table 3.1 presents the list of behavioural biometrics companies alongside a corresponding website where their script can be found.

We know that all of these websites contain the script associated with the company. However, we do not know if the script is active or not. To test it, we modified FoxHound

Behavioural Biometrics Company	Website	Flow Found
Ping Identity	Victoria Secret	Yes
ThreatMark	Sparkasse	Yes
Nudata	H&R Block	Yes
Neuro-Id	BILT Rewards	No
Iovation	Western Union	No
BehavioSec	Royal Bank	No
BioCatch	BMO	No
Accertify	Starbucks	No
CallSign	Wise	No
TypingDNA	TypingDNA	No
Transmit Security	Ethos Life	No
Forter	Citizen Watch	No
Sardine.ai	Wealthsimple	No
f5	CRA	No
Human	Fanduel	No
Darwinium	Asaas	No
Feedzai	Cajamar	No
DataVisor	AirAsia	No
BureauID	BureauID	No

Table 3.1: Taint flows caught by FoxHound without our changes

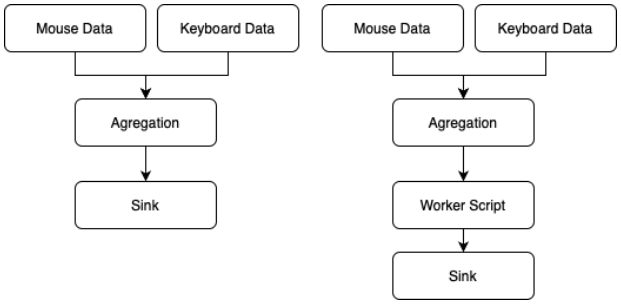


Figure 3.2: Typical flows for behavioural biometrics scripts

to catch behavioural biometrics flows. We added all mouse and keyboard events as sources. We kept the sinks the same as FP-Tracer [12] since it is the standard in sinks for data exfiltration. For each website, we visited the webpage, clicked on random places, clicked on form fields, inputted data into form fields, submitted forms and moved the mouse for a minute. We then looked at what taint flows were produced to see if FoxHound caught the behavioural biometrics scripts. Table 3.1 shows the result when running this version of FoxHound.

Only three companies produced taint flows out of the 19 we have. These results indicate that FoxHound is not in a state to be used for our analysis. We investigated every website that did not produce a flow. We wanted to know for each website if it was sending a flow and FoxHound was losing the flow or if the flow was never sent. To do this, we visited each website manually and analyzed the script. For each website where FoxHound was losing the taint flow, we went and tried to fix it. We will now discuss what insights we gained from analyzing the scripts and the fixes we brought to FoxHound.

In Figure 3.2, two common behavioural biometrics script flows are illustrated. The first flow processes behavioural data from the mouse and keyboard, aggregating it through mechanisms specific to the company. Once a predefined threshold is met—often determined by a specific amount of time elapsed or the volume of data collected—the aggregated data is sent to a sink. The sink, also company-specific, typically represents a method for extracting the data from the website. The second flow extends the first by incorporating a worker script, which handles computationally intensive tasks while the main script focuses on data collection.

3.4.2 Neuro-Id

The Neuro-ID script, which is used on the BILT Rewards website, initially evaded FoxHound's taint-tracking capabilities. This script processes mouse and keyboard event data, aggregates it through mathematical operations, and subsequently sends the aggregated data over the network. Upon investigating the taint flow, we discovered that the taint was being lost during the mathematical operations.

In JavaScript, mathematical operations are executed by the built-in `Math` library. As a built-in library, `Math` functions are implemented within the JavaScript engine, specifically in the `JSMath.cpp` file. Since this is part of the engine's internal code rather than standard JavaScript code, taint propagation does not occur automatically. Instead, it must be explicitly defined for each method in the library, which explains why the taint was being lost.

To address this issue, we modified the built-in `Math` library code to explicitly propagate taint metadata for all mathematical operations listed in the MDN Web Docs [39]. This ensured that taint tracking was preserved across operations like addition, subtraction, multiplication, and division, among others.

After implementing these changes, we tested the BILT Rewards website again using the updated version of FoxHound. This time, we successfully captured the taint flow, confirming that the modifications resolved the issue.

3.4.3 Iovation

While analyzing the Western Union website, where the Iovation script resides, we discovered that not all behavioural biometrics data was being propagated with the taint. Both keyboard and mouse events have multiple properties, primarily strings, numbers, and booleans. All these properties need to be taint-aware for effective tracking, but we found that FoxHound only supported taint propagation for strings and numbers. Boolean properties, which are also utilized by behavioural biometrics scripts, were not taint-aware and thus could not be tracked.

To address this, we implemented boolean tainting. Below, we outline the design decisions and implementation details involved in making booleans taint-aware.

Representation of Booleans in SpiderMonkey

JavaScript provides two types of booleans: primitive booleans and boolean objects. In the SpiderMonkey engine, both representations are handled in the `Boolean.cpp` file. This class wraps the boolean primitive, encapsulating the boolean value and a prototype pointer. Primitive booleans in JavaScript have their prototype set to `null`, while boolean objects use the boolean prototype.

To make booleans taint-aware, we introduced a new attribute to store taint information. For untainted booleans, this attribute remains `null`. When a boolean becomes tainted, the attribute is populated with the associated taint metadata.

Modifications for Taint Propagation

To access taint information from a tainted boolean in JavaScript (outside the SpiderMonkey engine), we added a new method to the boolean prototype that returns parsed taint data. Consequently, tainted booleans always have a prototype, which conflicted with SpiderMonkey's distinction between primitives (prototype `null`) and objects. To resolve this, we adjusted the engine to treat tainted booleans as primitives while maintaining their taint metadata. This trick ensures that tainting booleans does not disrupt the normal behaviour of JavaScript code.

Garbage Collection Adjustments

A significant issue arose with memory management: the garbage collector failed to remove taint metadata when the boolean object was deleted, causing memory bloat. To address this, we extended SpiderMonkey's nursery (its garbage collector) to ensure that taint information is cleaned up alongside the associated boolean object.

Comprehensive Taint Propagation

Next, we ensured proper taint propagation across all scenarios where booleans are used. This includes:

- Creating new objects from tainted booleans (e.g., strings or numbers).
- Creating booleans from tainted Objects.

- Converting tainted booleans to other object Tainted types.
- Converting tainted Numbers to other tainted booleans.
- Converting tainted Strings to other tainted booleans.
- Stringifying tainted booleans using JSON methods.

We reviewed all potential operations and added logic to propagate taint as required. Additionally, we incorporated taint checks and propagation into SpiderMonkey's interpreter. The interpreter evaluates JavaScript code by parsing it into an abstract syntax tree (AST), compiling it into bytecode, and interpreting it sequentially using a stack-based architecture. Whenever an operation involved a tainted boolean, we ensured the resulting value inherited the taint before being pushed back onto the stack.

Documentation and Testing

Given the extensive changes made to the SpiderMonkey engine, we added detailed documentation explaining the boolean tainting implementation. Furthermore, we developed extensive unit tests to verify proper taint propagation in various scenarios.

Results

After implementing boolean tainting, we revisited the Western Union website. The taint flows now correctly include all properties of the keyboard and mouse events, including boolean properties, enabling comprehensive tracking of behavioural biometrics data by FoxHound.

3.4.4 BehavioSec

Royal Bank uses the BehavioSec behavioural biometrics script. However, when looking at the taint flows we noticed that they were not caught by FoxHound. After diving deeper into the script, we saw that it takes mouse and keyboard data and aggregates them all together in a JSON. It then sends that JSON to a message port, which we consider to be a sink. That JSON holds all the behavioural biometrics data and it is tainted right before it goes to the sink. This tells us that the taint is being lost right before reporting the taint flow.

When looking at the FoxHound code, we realized that almost all the sinks take in strings that contain the taint flow information. So if a JSON is passed to a sink, it will be transformed to a string before being passed to the sink. All the taint data from the JSON will propagate to the string when it is stringified. However, not all sinks work in this manner. Some sinks do not implicitly transform what is passed to them into a string. Some sinks will take directly what is passed to them and since the reporting of a sink is based on a string, the JSON will be missed and the taint flow will not be properly reported. We added the necessary code that recursively loops over objects that are not strings and properly reports all sub-elements if they are tainted. With this fix to the reporting sinks code, we were able to catch the taint flows for the BehavioSec script.

3.4.5 BioCatch

We looked at the BMO websites for the Biocatch script. We were not able to catch the taint flow from sources to sinks. We then proceeded to look at the code and we discovered that it was taking behavioural biometrics data, aggregating it together and sending it out of the website through network requests. We further investigated the script and found out that BioCatch uses worker scripts to facilitate the workload on the main script and do data manipulation in parallel. We noticed that when the tainted data leaves the main script and goes to the worker script, the taint information is lost.

To better understand the issue, we need to first understand why JavaScript uses worker scripts. JavaScript is single-threaded by nature [39], which is a decision choice that was made to keep the language simple and avoid complicated resource distribution scenarios. However, some tasks are blocked and those tasks will make the script slow down. This is why JavaScript has asynchronous methods and callback functions. Sometimes, these solutions are not enough and more separation is needed. JavaScript offers web workers, which are simple scripts that can be spanned from the main thread and are used to run scripts in background threads [39]. Since JavaScript is single-threaded, the way these web workers are implemented is by spanning a new instance of JavaScript, which means there will be two completely separate JavaScript scripts running. One will be called main, which will run on the main thread and the other one will be the worker script that will run on background threads. These two scripts are independent of each other and have their own context. However, the main script will need to communicate with the worker script to pass data. Since these two scripts are their own processes, the communication is done via inter-process communication (IPC).

In SpiderMonkey, inter-process communication between the main script and worker scripts is implemented using the Structured Clone algorithm. This algorithm allows

JavaScript objects to be serialized and passed between threads while maintaining their integrity and structure. When a message is sent from the main script to a worker, SpiderMonkey serializes the object into a format that can be transmitted across thread boundaries. The worker script then deserializes the object upon receiving it, reconstructing the original data structure in its local context.

The serialization process, however, has limitations. Certain types of data, such as functions or objects with complex prototypes, cannot be transferred using the Structured Clone algorithm. This mechanism of serialization and deserialization ensures that no direct references are shared between the main script and the worker script, maintaining thread safety. However, this also means that any metadata, such as taint information applied to the data in the main script, is not inherently preserved during the cloning process. This behaviour is by design since the Structured Clone algorithm focuses on data fidelity rather than preserving auxiliary metadata that might not conform to standard JavaScript data types.

In the context of FoxHound and the BioCatch script, this limitation becomes a critical challenge. When tainted data is sent from the main script to a worker script, the serialization process strips the data of its tainted information. This results in the worker script processing the data as if it were untainted. The loss of taint information occurs because the taint metadata is not part of the core data structure but rather an external annotation that the Structured Clone algorithm does not account for.

To address this issue, we modified the serialization and deserialization process in SpiderMonkey to explicitly handle taint metadata for numbers and booleans. By extending the Structured Clone algorithm, we were able to embed taint information directly into the serialized representation of these types. During deserialization, the worker script reconstructs not only the original value but also its associated taint metadata, ensuring that the taint is not lost. With this enhancement, FoxHound can now reliably track the flow of tainted numbers and booleans between the main script and worker scripts, which allowed us to catch the BioCatch script.

3.4.6 Accertify

The Accertify script was found on the Starbucks website. Upon further analysis of the Accertify script, we observed that taint information was lost when the `btoa` method was called. The `btoa` method in JavaScript creates a Base64-encoded ASCII string from a binary string [39]. To investigate further, we attempted to reproduce this behaviour on

our website, but the taint flow remained intact when using the `btoa` method. This indicated that there was another factor in the Accertify script contributing to the issue.

Upon deeper analysis, we discovered that the `btoa` method was being invoked on an instance variable of a class defined by the script. To validate this, we created a custom class and called the `btoa` method on one of its instance variables, successfully reproducing the bug.

We isolated the issue and identified it as being specific to the use of classes in conjunction with the `btoa` method. A closer examination of the FoxHound code revealed that a fast path was being used for string conversion in this context. Ordinarily, the conversion from a binary string to an ASCII string occurs in a section of the code that is taint-aware. However, when classes and instance variables are involved, the code reuses memory allocated for the resulting string. This optimization, enabled by the use of classes, allows the code to exit via the fast path without propagating taint information.

To address this, we modified the fast path to ensure it is taint-aware. Additionally, we reviewed and updated other fast paths in the codebase that were not taint-aware. These changes ensured that taint information was properly propagated, even in optimized execution paths.

After implementing these changes, the taint flow was no longer lost during the `btoa` method call. We were able to successfully track taint flow throughout the entire code, from sources to sinks, ensuring the integrity of the taint-tracking system.

3.4.7 CallSign

On the Wise website, we discovered a CallSign script that performs behavioural biometrics. However, we were initially unable to track the taint flow within this script. Upon further investigation, we observed that CallSign utilizes typed arrays—a specialized array type in JavaScript that differs from standard arrays.

In JavaScript, standard arrays are dynamic, meaning their size and the memory allocated for each element can vary. In contrast, typed arrays have a fixed size in memory, ensuring that each element occupies a specific number of bytes. For example, a `Uint8Array` is a typed array where each slot contains an 8-bit unsigned integer [39]. Typed arrays are typically used to preprocess data, particularly in scenarios where data needs to be shared across services or processes. Their fixed memory structure allows for efficient referencing of a shared buffer without the risk of changes to the number or size of slots.

The fixed size of each slot in typed arrays presented a challenge for taint propagation. Taint metadata can vary in size based on the operations performed on a tainted value, which is incompatible with the rigid structure of typed arrays. Initially, we explored the possibility of making typed arrays taint-aware by embedding taint information directly into their structure. However, this approach disrupted the ability to share data between processes and services, a core functionality of JavaScript.

To address this limitation without compromising JavaScript's core behaviour, we sought an alternative solution. Through our analysis of typed arrays and their role in behavioural biometrics scripts, we found that their primary use case is to preprocess data before exfiltrating it. Based on this observation, we determined that it would be appropriate to designate typed arrays as sinks for tainted data. By marking them as sinks, any data written to a typed array would be considered exfiltrated, allowing us to track taint flow effectively. We did not see typed arrays being used for other purposes that were not related to data preprocessing.

We implemented this solution in the FoxHound code by modifying it to treat typed arrays as sinks. This required accounting for all the different ways typed arrays can be created, ensuring comprehensive coverage.

After implementing these changes, we revisited the Wise website and successfully tracked the taint flow from behavioural biometrics data to the typed arrays. This enhancement allows FoxHound to effectively monitor and analyze the use of typed arrays in similar scripts, ensuring robust taint tracking in behavioural biometrics scenarios.

3.4.8 TypingDNA

On the TypingDNA website, a demo showcases their product, which involves typing a predefined text to generate a unique typing pattern. Once the text is completed and the form submitted, the typing pattern is transmitted to TypingDNA's back end for behavioural biometrics analysis. During our investigation, we observed that the taint flow from keyboard events to the storage of the typing pattern was being lost. Upon analyzing the TypingDNA code, we discovered that only the first key press event was retained, while subsequent key press events were discarded. Further examination revealed that the taint loss originated from a bug in FoxHound.

To better understand the issue, we reproduced the bug in an isolated environment. We found that the taint was successfully tracked for the first key press but was lost for subsequent ones. By tracing the bug, we identified the root cause as a caching issue. Specifically, when a key was pressed, Firefox stored an inline cache for future reference. Inline caches

are utilized by the Just-In-Time (JIT) and Ion compilers to optimize JavaScript execution. However, neither of these compilers is taint-aware, resulting in the loss of taint information. In contrast, the baseline JavaScript interpreter in Firefox is taint-aware. When inline caches were accessed, the code execution path entered the JIT compiler, causing the taint to be lost.

Initially, we attempted to disable the JIT and Ion compilers through Firefox's configuration settings, but the compilers continued to be utilized despite being turned off. To address this issue, we implemented a solution that prevented the creation and attachment of inline caches altogether. This approach forced the code to always take the slow path through the taint-aware interpreter, thereby preserving taint information.

After applying these changes, we re-tested TypingDNA and observed that while the original taint loss point was resolved, the taint flow was still being lost later in the code. Specifically, before the typing pattern was sent over the network, it was added to a value in a hidden form element. FoxHound already classifies form fields as sinks, but TypingDNA accessed the form field using a unique method that was not covered by the existing sink definitions in FoxHound. The API it was using was called `HTMLInputElement`.

To address this oversight, we updated FoxHound to include this specific method of accessing form fields as a sink. With these enhancements, we re-ran TypingDNA and successfully tracked the taint flow from the keyboard events to the form field and eventually to the network transmission. These improvements ensure that FoxHound can effectively monitor and analyze taint flows in similar scenarios.

3.4.9 Transmit Security

The Transmit Security script can be found on Ethos Life. Upon visiting the website, we observed that it utilizes the `TextEncoder` built-in class from JavaScript [39]. The `TextEncoder` class provides two methods: `encode` and `encodeInto`. Both methods accept a string as input, convert it into a UTF-8 encoded representation, and store the result in a `Uint8Array`. However, we noticed that taint information was being lost when the `encode` method was called on a tainted string.

Based on the changes previously implemented, `Uint8Array` instances are already marked as sinks since they are a type of typed array. To understand why taint was not propagating specifically in the case of `TextEncoder`, we conducted further investigation.

We discovered that both the `encode` and `encodeInto` methods perform an intermediate transformation, converting the tainted string into a UTF-8 representation. This

transformation is a unique process that does not rely on the standard string conversion mechanisms, which are already taint-aware. As a result, taint metadata was not preserved during this special conversion.

To resolve this issue, we modified the implementation of the UTF-8 transformation in `TextEncoder` to make it taint-aware. This involved ensuring that the taint metadata from the input string is propagated correctly to the resulting `Uint8Array`. Once this change was implemented, taint tracking for the `encode` and `encodeInto` methods was restored.

After making these updates, we re-evaluated the Ethos Life website and confirmed that the taint flow could now be properly tracked from sources to sinks within the Transmit Security script. This enhancement ensures that taint propagation remains consistent, even in cases involving special string transformations.

3.4.10 Forter

The Forter script is employed on the Citizen Watch website. During our initial analysis using the unmodified version of FoxHound, we were unable to detect any taint flows reaching sinks. However, after implementing the modifications developed for other scripts, we observed some tainted flows successfully reaching sinks during the initial execution of the script. Despite this progress, the taint flow ceased to appear after a few seconds, prompting further investigation into why FoxHound could no longer track the flows.

The Forter script is heavily obfuscated, making it challenging to fully understand its behaviour. From our analysis, we determined that it captures keyboard and mouse events. Periodically, it initiates network requests to Forter's back end, which we believe are related to behavioural biometrics data. Interestingly, we observed a correlation between the volume of user interactions and the network activity: as more keyboard and mouse events occur, the frequency of network requests increases, and the size of the transmitted data grows. This strongly suggests that the script aggregates behavioural data for transmission to its back end.

To pinpoint where the taint flow was being lost, we delved deeper into the script's operation. We discovered that the script processes tainted values through a recursive function with two branching paths and a maximum depth of five. At each step of the recursion, the function decides what path to take by comparing the tainted value to another value, the purpose of which remains unclear due to the obfuscation. The recursive process culminates in a non-tainted value being returned, depending on the chosen path.

From our analysis, it appears that this recursive function operates as a decision tree. The function's output is influenced by behavioural biometrics data (the tainted values),

but the returned result itself is not tainted. To enable FoxHound to track this flow, we would need to implement conditional branch taint propagation. However, this is not a viable solution for several reasons.

Typically, not every operation or value within a conditional branch should be tainted simply because the condition itself is tainted. Introducing such broad taint propagation would risk diluting FoxHound's fine-grained tainting guarantees, potentially leading to over-tainting and false positives. These guarantees are critical to accurately identifying true taint flows without conflating unrelated data.

Due to this inherent limitation, FoxHound is unable to track the taint flow through this decision tree. While the behaviour highlights an edge case in taint tracking, addressing it would require a fundamental trade-off that could compromise FoxHound's overall precision.

With a clearer understanding of Forter's functionality, we classified its behaviour in terms of FoxHound's effectiveness. While FoxHound can successfully track the initial tainted flows sent over the network by Forter, the taint is lost when the script transitions to its decision-based model. As a result, subsequent flows derived from the decision tree are not tracked. For this reason, we classify the Forter script as "half caught", meaning that FoxHound can only capture the taint flows for the initial phase of the script's execution.

3.4.11 Sardine.ai, f5, Human

Sardine.ai, f5 and human are three scripts that we were not able to catch the taint flows with the original version of FoxHound. However, after integrating the modifications developed for other scripts, FoxHound successfully captured the taint flows in these cases without requiring any additional changes. This outcome demonstrates that the foundational improvements made to FoxHound have significantly enhanced its ability to track taint flows across diverse scripts. While it is not possible to guarantee that all instances of taint loss have been identified, we believe that the most critical edge cases have been thoroughly addressed.

3.4.12 Darwinium, Feedzai, DataVisor, BureauID

For Darwinium, Feedzai, DataVisor, and BureauID, FoxHound did not identify any taint flows, but our analysis suggests this is due to the absence of taint flows rather than a limitation of FoxHound.

Darwinium and BureauID are not active on the websites where they were found. This indicates that the script is embedded on the website but is not invoked to process any data, a behaviour that can be achieved through the use of feature flags. Without behavioural biometrics information being sent out, these scripts do not generate taint flows. Their inactivity aligns with our findings, affirming that no modifications to FoxHound are required.

Feedzai presents a unique challenge due to its heavy obfuscation, making it difficult to determine its exact functionality. The script periodically sends network requests, listens to mouse and keyboard events, and aligns with Feedzai's advertised behavioural biometrics capabilities. However, during testing, the network requests were consistently empty in both the modified FoxHound browser and a standard unmodified browser. This behaviour indicates that the absence of taint flows is not caused by a limitation in FoxHound. We hypothesize that Feedzai only transmits behavioural biometrics data when the user is logged into their account. Unfortunately, as we lack access to an account on the website employing this script, we cannot confirm this hypothesis. For now, there is no evidence of taint flow to be captured.

DataVisor is a modular script with multiple functionalities, activated based on user requirements. The functionality relevant to behavioural biometrics is known as DataVisor Edge. On the website where we encountered DataVisor, the Edge functionality was not enabled. While the script may collect behavioural biometrics data, this data remains confined to the client side without being transmitted via a sink. This scenario highlights the advantage of FoxHound's dynamic approach over static analysis. FoxHound effectively determines whether a script genuinely transmits behavioural biometrics data or merely processes it locally.

3.4.13 Modified FoxHound

We have compiled all the modifications made to FoxHound and this new and improved version can be found in our GitHub repository [9]. These changes are currently in the process of being integrated into the official FoxHound repository. We also re-ran FoxHound on all the websites we had from our original Table 3.1. The new results can be found in Table 3.2

behavioural Biometrics Company	Website	Flow Found
Ping Identity	Victoria Secret	Yes
ThreatMark	Sparkasse	Yes
Nudata	H&R Block	Yes
Neuro-Id	BILT Rewards	Yes
Iovation	Western Union	Yes
BehavioSec	Royal Bank	Yes
BioCatch	BMO	Yes
Accertify	Starbucks	Yes
CallSign	Wise	Yes
TypingDNA	TypingDNA	Yes
Transmit Security	Ethos Life	Half
Forter	Citizen Watch	No
Sardine.ai	Wealthsimple	Yes
f5	CRA	Yes
Human	Fanduel	Yes
Darwinium	Asaas	N/A
Feedzai	Cajamar	N/A
DataVisor	AirAsia	N/A
BureauID	BureauID	N/A

Table 3.2: Taint flows caught by FoxHound with our changes

3.5 Discussion

When we first looked at the results from crawling the 19 behavioural biometrics websites with the unmodified version of FoxHound, we only found three taint flows that went from sources to sinks. After analyzing all the 19 scripts we were able to better understand how they work. This is also true for the heavily obfuscated scripts, which took a significant amount of time to extract information from. From the 19 scripts, we were able to determine that only 15 of them were active. The other four were either inactive or were not sending out behavioural biometrics data. This means that FoxHound only needs to catch 15 flows instead of 19 as previously thought.

3.5.1 Effectiveness Improvements

We measured the effectiveness of FoxHound by evaluating its ability to track taint flows in behavioural biometrics scripts. Effectiveness is defined as the percentage of scripts for which at least one taint flow was successfully tracked, relative to the total number of scripts that actively send behavioural biometrics data.

- Before Modifications: FoxHound successfully tracked taint flows in 3 out of 15 active scripts, achieving an effectiveness rate of 20%.
- After Modifications: FoxHound successfully tracked taint flows in 14 out of 15 active scripts, resulting in an effectiveness rate of 97%.

This represents an almost fivefold improvement in effectiveness, showing the impact of the enhancements made to FoxHound.

3.5.2 Observations

FoxHound has been evaluated using non-minified and non-obfuscated scripts [12, 32]. Testing with standard scripts ensures that taint propagation functions correctly throughout the code, demonstrating FoxHound's capability to handle normal scripts effectively. However, obfuscated scripts have not been extensively tested due to the inherent difficulty in understanding their functionality.

In such cases, taint propagation may be incomplete or lost without significant consequences, as the behaviour of the obfuscated script is not fully understood. It remains unclear whether the script should have produced specific taint flows or not.

```
V=Lt(a,N),P=M(X(p));if(e[X(m)]=V&&!!V[P],e[X(y)]=function(){var t=434,e=434,n=651,r=623,o=455,i=668,c=uh;try{var u=a[c(t)]&&a[c(e)][c(n)];if(u)return
ra!==u[c(r)]||aa!==u[c(o)]||oa!==u[c(i)]}catch(t){}}(),e[X(g)]=function(){var t=631,e=432,n=uh;try{(void 0)[n(t)]}catch(t){return t[n(e)]()}}(),e[X(E)]=function(){var
t=485,e=729,n=740,r=690,i=744,c=797,u=497,s=uh;try{return Array[s(t)][s(e)][s(n)](a[s(r)](o[s(i)],""))[s(c)]("")[s(u)](/-(moz|webkit|ms)-/)[1]}catch(t){}}(),e[X(b)]=function(){var
t=534,e=432,n=413,r=uh;try{return a[r(t)][r(e)]()[r(n)]}catch(t){}}(),e[X(T)]=(n=532,/constructor/i[(r=uh)(438)](a[r(n)])),e[X(S)]=function(){var t=701,e=539,n=432,r=478,o=uh;try{var
i=a[o(t)]&&a[o(t)][o(e)];if(i)return i[o(n)]()===M(o(r))}catch(t){}}(),e[X(I)]=function(){var e=663,n=709,r=663,a=546,o=438,i=710,c=717,u=663,s=413,l=663,v=uh,p=!1;try{p=(typeof global===v(e)?
v(e):t(global))===d&&String(global)===v(n)}catch(t){}try{p=p||(typeof process===v(r)?v(r):t(process))===d&&String(process)===v(a)}catch(t){}try{p=p||!0===/node|io\.js/[v(o)](process[v(i)]
[v(c)])}catch(t){}try{p=p||(typeof setImmediate===v(u)?v(e):t(setImmediate))===h&&4===setImmediate[v(s)]}catch(t){}try{p=p||(typeof __dirname===v(l)?v(r):t(__dirname))===f}catch(t){}return p}
(),e[X(w)]=function(){var t={d:720},e=uh;try{var n=M(e(t.d));new Worker(n);return!0}catch(t){return!1}}(),e[X(O)]=function(){var t=613,e=464,n=667,r=797,o=770,i=438,c=612,u=uh;try{return
Object[u(t)](a)[u(e)]((function(t){var e=u;return/^(s|a).*(usc|da).*/[e(i)](t[e(c)]())}))[u(n)]()[u(r)](".")[u(o)](0,100)}catch(t){}}(),Ha){var
_=M(X(R)),U=M(X(A)),F=M(X(k));e[X(x)]=ee(N,_),e[X(B)]=ee(N,U),e[X(C)]=ee(N,F)}catch(t){}}function lh(t,e,n){var r,a=!1,o=(r=new Blob([t],
{type:"application/javascript"}),URL.createObjectURL(r)),i=new Worker(o);return i.onmessage=function(t){return e(t)},i.onerror=function(t){if(!a)return a=!0,function(t,e){try{return t()}catch(t)
{if(e)return t}}((function(){i.terminate()})),n(t)},i}function fh(t,e){var n=hh();return(fh=function(t,e){return n[t-=318]})(t,e)}function hh(){var t=
["3465371cUArOj","oDeoeLy","reverse","3611368HKBFSu","1067470qWUzha","JVcGF","split","join","jMT","3478460eDcLmM","ppErxl","floor","EJhO","838351FtDdNX","apply","1965381Cgwudr","6vDmmDn","138514
80FnQhXN","GT38z","NktHDHMmRj0=","i4Ld6PDHrx","VE1q","TVl"];return(hh=function(){return t})()}function dh(){var t=324,e=333,n=fh;try{if(yh(n(t)))gh(function(d,a,m){return(d*9350)-
m.charCodeAt(14)}[n(e)](v,Hf))}catch(t){}}function vh(){var t=331,e=333,n=fh;try{if(yh(n(t)))gh(function(v,y,p){return(v*1417)*p.charCodeAt(8)}[n(e)](v,Hf))}catch(t){}}function ph(){var
t=337,e=333,n=fh;try{if(yh(n(t)))gh(function(s,l,g){return(s+5852)+g.charCodeAt(7)}[n(e)](v,Hf))}catch(t){}}function mh(){var t=fh;try{if(yh("k1"))gh(function(a,p,t)
{return(a*7464)*p.charCodeAt(14)}[t(333)](v,Hf))}catch(t){}}function yh(t){return Gf===t}function gh(t){var e=338,n=330,r=fh;!Wf[r(e)]&&(Wf[r(e)]=Zt(""+Math[r(n)](t)))}function Eh(){var
t=339,e=333,n=fh;try{if(yh(n(t)))gh(function(z,h,b){return Math.floor(z/8890)/b.charCodeAt(25)}[n(e)](v,Hf))}catch(t){}}function bh(t){var e=fh;try{Wf=t,Hf=[Ea,vt(),Kr()],Gf=function(t){var
e=325,n=321,r=326,a=fh;return M(t)[a(e)]("")[a(n)]()[a(r)]("")}(e(340)),dh(),Sh(),dh(),Th(),ph(),vh(),mh(),Ih(),Sh(),Eh(),ph(),Ih(),mh(),Eh(),vh(),Oh(),wh(),Oh(),wh(),Th()}catch(t){}}function
Th(){var t=318,e=333,n=fh;try{if(yh(n(t)))gh(function(b,d,s){return(b-4975)-d.charCodeAt(18)}[n(e)](v,Hf))}catch(t){}}function Sh(){var t=327,e=333,n=fh;try{if(yh(n(t)))gh(function(t,c,k){return
Math.floor(t/1655)/c.charCodeAt(19)}[n(e)](v,Hf))}catch(t){}}function Ih(){var t=fh;try{if(yh("Kh"))gh(function(n,l,p){return(n-5216)-l.charCodeAt(32)}[t(333)](v,Hf))}catch(t){}}function wh()
{var t=320,e=333,n=fh;try{if(yh(n(t)))gh(function(a,e,s){return(a+2818)+e.charCodeAt(1)}[n(e)](v,Hf))}catch(t){}}function Oh(){var t=329,e=333,n=fh;try{if(yh(n(t)))gh(function(h,t,w){return
Math.floor(h/6404)*t.charCodeAt(26)}[n(e)](v,Hf))}catch(t){}}!function(t,e){for(var n=461,r=605,a=785,o=607,i=754,c=581,u=782,s=644,l=509,f=577,h=uh,d=t();;)try{if(965478===-parseInt(h(n))/1*
(parseInt(h(r))/2)+-parseInt(h(a))/3+parseInt(h(o))/4+parseInt(h(i))/5*(-parseInt(h(c))/6)+-parseInt(h(u))/7*(-parseInt(h(s))/8)+-
parseInt(h(l))/9+parseInt(h(f))/10)break;d.push(d.shift())}catch(t){d.push(d.shift())}}(oh),function(t,e){for(var n=332,r=323,a=334,o=322,i=328,c=335,u=319,s=336,l=fh,f=t();;)try{if(483789===-
parseInt(l(n))/1+parseInt(l(r))/2+-parseInt(l(a))/3+parseInt(l(o))/4+-parseInt(l(i))/5+-parseInt(l(c))/6*(parseInt(l(u))/7)+parseInt(l(s))/8)break;f.push(f.shift())}catch(t){f.push(f.shift())}}
(hh);var Rh=Kh;!function(t,e){for(var n=306,r=366,a=523,o=388,i=368,c=486,u=350,s=430,l=192,f=152,h=140,d=Kh,v=t();;)try{if(129453===-parseInt(d(n))/1*(parseInt(d(r))/2)+-parseInt(d(a))/3*(-
parseInt(d(o))/4)+-parseInt(d(i))/5*(parseInt(d(c))/6)+parseInt(d(u))/7+-parseInt(d(s))/8+-parseInt(d(l))/9*(parseInt(d(f))/10)+parseInt(d(h))/11)break;v.push(v.shift())}catch(t)
{v.push(v.shift())}}(Gh);var Ah,kh,xh,Mh={},Bh=
[Rh(174),Rh(202),Rh(506),Rh(353),Rh(329),Rh(457),Rh(463),Rh(522),Rh(478),Rh(396),Rh(319),Rh(298),Rh(275),Rh(496),Rh(331),Rh(414),Rh(303),Rh(225),Rh(272),Rh(355),Rh(390),Rh(345),Rh(524),Rh(419),R
h(288)],Ch=M(Rh(435)),Xh=M(Rh(149)),Nh=M(Rh(401)),Vh=M(Rh(512)),Ph=[Ch,Xh,Nh],_h=Rh(424),Uh=30;function Fh(e){var
n=470,r=482,o=507,c=209,s=324,l=193,f=270,v=378,p=470,m=228,y=289,g=470,E=318,b=478,T=285,S=396,I=159,w=298,O=456,R=319,A=214,k=275,x=258,M=337,B=258,C=414,X=496,N=233,V=521,P=456,_=222,U=191,F=
394,W=230,H=325,G=271,D=383,L=247,Z=481,Y=520,Q=212,j=334,J=176,z=354,K=307,q=426,$=327,tt=238,et=428,nt=412,rt=142,at=412,ot=467,it=251,ct=178,ut=251,st=183,lt=371,ft=251,ht=224,dt=155,vt=251,p
t=199,mt=340,yt=251,gt=399,Et=445,bt=210,Tt=162,St=243,It=198,wt=170,Ot=500,Rt=479,At=203,kt=459,xt=441,Mt=440,Bt=145,Ct=448,Xt=429,Nt=246,Vt=484,Pt=332,_t=425,Ut=297,Ft=415,Wt=431,Ht=226,Gt=320
,Dt=148,Lt=154,Zt=331,Yt=240,Qt=489,jt=196,zt=358,Kt=Rh,qt=!1,$t=-1,te=[];i[Kt(n)]&&(qt=function(){var e,n=
{y:470,L:229,E:470,Q:229,I:470,O:446,P:470,K:446,h:229,H:470,l:433,M:408,V:476},r=Rh;if(!i[r(n.y)])return!1;e=t(i[r(n.y)][r(n.L)])===h?i[r(n.E)][r(n.Q)]():i[r(n.I)][r(n.O)]&&t(i[r(n.P)][r(n.K)]
[r(n.h)])===h?i[r(n.E)][r(n.K)][r(n.Q)]():t(i[r(n.H)]);return e===r(n.l)||e===r(n.M)||e===r(n.V)}(),$t=i[Kt(n)][Kt(r)],te=function(){var t={y:470,L:482,E:311,Q:470,I:467},e=Rh,n=[];try{for(var
r=0;r<i[e(t.y)][e(t.L)]&&r<Uh;r++)n[e(t.E)](i[e(t.Q)][r][e(t.I)])}catch(t){}return n}(),e[Kt(o)]=te,e[Kt(c)]=$t,e[Kt(s)]=e[Kt(l)]=qt,e[Kt(f)]=ea;try{e[Kt(v)]=i[Kt(n)][0]===i[Kt(p)][0][0]
[Kt(m)]}catch(t){}try{e[Kt(y)]=i[Kt(g)][Kt(E)](4294967296)===i[Kt(g)][0]}catch(t){}try{e[Kt(b)]=i[Kt(T)],e[Kt(S)]=i[Kt(I)],e[Kt(w)]=i[Kt(O)],e[Kt(R)]=i[Kt(A)],e[Kt(k)]=!!
(i[Kt(x)]||null===i[Kt(x)]||i[Kt(M)]||a[Kt(B)]),e[Kt(C)]=function(){var t={y:509},e=Rh;try{return(new Date)[e(t.y)]()}catch(t){return 9999}}(),e[Kt(X)]=i[Kt(N)],e[Kt(V)]=i[Kt(P)]&&i[Kt(P)]
[Kt(r)]}catch(t){}try{t(i[Kt(_)])!==d&&!i[Kt(_)]&&(e[Kt(U)]=u),e[Kt(F)]=i[Kt(W)],e[Kt(H)]=i[Kt(G)],e[Kt(D)]=i[Kt(L)],e[Kt(Z)]=e[Kt(Y)]=function(){var t={y:334,L:229,E:153,Q:389},e=Rh;try{var
n=i[e(t.y)]&&i[e(t.y)][e(t.L)]();return n===e(t.E)||/MSMimeTypesCollection/i[e(t.Q)](n)}catch(t){return!1}}(),e[Kt(Q)]=i[Kt(j)]&&i[Kt(j)][Kt(r)]||-1}catch(t){}try{e[Kt(J)]=i[Kt(z)]}catch(t)
{}try{e[Kt(K)]=i[Kt(q)]}catch(t){}try{e[Kt($)]=i[Kt(tt)]}catch(t){}try{e[Kt(et)]=i[Kt(nt)]&&i[Kt(nt)][Kt(rt)]&&i[Kt(at)][Kt(rt)][Kt(ot)]===Kt(rt)}catch(t){}try{i[Kt(it)]&&(e[Kt(ct)]=i[Kt(ut)]
[Kt(st)],e[Kt(lt)]=i[Kt(ft)][Kt(ht)],e[Kt(dt)]=i[Kt(vt)][Kt(pt)],e[Kt(mt)]=i[Kt(yt)][Kt(gt)])}catch(t){}try{e[Kt(Et)]=Kt(bt)in
i&&!0===i[Kt(bt)],e[Kt(Tt)]=i[Kt(_)]+""===Kt(St),e[Kt(It)]=!!Cn(),Ha&&(e[Kt(wt)]=Kt(Ot)in i&&!0===i[Kt(Ot)])}catch(t){}na&&
(e[Kt(Rt)]=na[Kt(At)],e[Kt(kt)]=na[Kt(xt)],e[Kt(Mt)]=na[Kt(Bt)],e[Kt(Ct)]=na[Kt(Xt)],e[Kt(Nt)]=na[Kt(Vt)],e[Kt(Pt)]=na[Kt(I)],e[Kt(_t)]=na[Kt(Ut)],e[Kt(Ft)]=na[Kt(Wt)]);try{e[Kt(Ht)]=!!i[Kt(Gt)]
,e[Kt(Dt)]=i[Kt(Lt)]}catch(t){}oe(e,Kt(Zt),(function(){return i[Kt(zt)]}),-1);try{e[Kt(Yt)]=!Jt(i[Kt(Qt)][Kt(jt)])}catch(t){}}function Wh(t){var e=parseFloat(t);if(!isNaN(e))return e}function
Hh(e){if(t(e)!==u)return Zt(e)}function Gh(){var t=["orientation","SBl5Xg1yf2g=","Chc7UE95PWM=","Content Security
Policy","XGVtYhkLa1k=","visible","item","Bzp2fUFVdUs=","userAgentData","height","R3o2PQIWNAg=","__webdriver_script_fn","MDFBNnVaRgw=","bRRcEyt1Wik=","getOwnPropertyDescriptor","eWhILz8ESRQ=","cs
sFromStyleSheets","XGVtYhoOY1A=","outerHeight","DFU9Eko7MyU=","AWBwJ0QNdxQ=","offsetWidth","mimeTypes","UTAgdxRaJkE=","geb","msDoNotTrack","Ix4SGWZ0ES8=","width","eEkJDj4kCT0=","ajcbMC9aGAI=","g
etElementsByTagName","SBl5Xg50eGw=","FUxkS1MjYHo=","Az5yeUVTcU4=","FwpmDVJmZTs=","scrollY","localStorage","instantiate","249207gqoGMU","documentElement","split","T3I+NQoZMAA=","appName","Z1pWXSE
3VG8=","visibility","WidrIBxKZBU=","hardwareConcurrency","IUAQR2cuFnc=","FC0lKlJDIB4=","tagName","voiceURI","getAttribute","bWVtb3J5","aRhYHyxxWCQ=","2268nqqNun","outerWidth","14770jXlYyY","type
","ZjsXPCBXGAo=","OAlJTn5iSH4=","NA1FSnFgQXg=","awesomium","VQwkCxBgJjo=","TTQ8cwtbOkA=","elementFromPoint","input","GwZqAV1tZDc=","moz","LVQcU2s4Gmc=","HmMvZFgDLVE=","fontFromResourceApi","Ah8z
WEd0NG4=","TouchEvent","QSAwZwdNN1A=","Q34yOQUSMA0=","fEUNAjouAzc=","192ofMYmB","test","KndbcGwaXks=","shift","imgFromResourceApi","_cordovaNative","bHUdcikdE0E=","GwZqAV5sZTI=","EXBgN1QcYw0=","
battery","dWxEKzMDRh8=","effectiveType","callPhantom","bmF2aWdhdG9yLnVzZXJBZ2VudA==","Ql9zGAc0fSM=","DateTimeFormat","Date","eWhILz8DTh4=","BatteryManager","offsetHeight","[object
MSPluginsCollection]","HCUtIlpFKBM=","hidden","eyZKYT5LSls=","permissions","MVAAV3cwBWw=","Y15SWSU1XWs=","bRRcEyh5WyI=","U;2cQ.cVek%.~","Buffer","getBattery","fEUNAjkoCTc=","matches","ZH0VeiIRFU
g=","cookie","Z1pWXSI3UG4=","missing","U04iSRYjJXk=","buildID","AWBwJ0QNchI=","Ql9zGAQxci4=","mobile","128000lyCexq","uaFullVersion","call","[object
PluginArray]","BF01GkIwMCs=","bmF2aWdhdG9yLndlYmRyaXZlcg==","timing","webkit","GmcrYFwILVc=","DhM/VEh+O24=","cg8DSDdiBXI=","bitness","setItem","fEUNAjorDjE=","Worklet","LnNfdGgTXkE=","constructo
r","get","Bhs3XEN2MWc=","label","Y15SWSUzVGg=","availWidth","DFU9Ekk9OSM=","availHeight","SBl5Xg11fG0=","some","languages","Az5yeUZWdEg=","ontouchstart","RTw0ewBRMk4=","Ix4SGWVzFSw=","anNIZWFwU2
l6ZUxpbWl0","W0YqQR4uK3A=","S3Y6MQ4fPgs=","colorDepth","onorientationchange","2,10","name","substring","hasOwnProperty","plugins","external","In9TeGQSV0M=","FwpmDVFhZDY=","getEntries","BXx0O0MRc
As=","[object
HTMLPluginsCollection]","AWBwJ0cBcxw=","S3Y6MQ0bNAA=","HwJuBVpvaDE=","_Selenium_IDE_Recorder","Rlt3HAA3dyo=","length","bind","model","Yj8TOCdVEgs=","330lxMetz","cy5CaTZGRlo=","DXR8M0gceAg=","ser
viceWorker","KxYaEW12GiQ=","requestAnimationFrame","chrome","OkdLAH8uTjE=","showModalDialog","v8Locale","RB11WgJzcWk=","Sld7EA85fSE=","NktHDHArST0=","Ui9jKBdHZhw=","cookieEnabled","message","Bzp
2fUFbc0c=","OkdLAHwoTjo=","aRhYHy91WiU=","ZRxUGyB3VS4=","GmcrYF8OL1M=","Y15SWSY2UmI=","dWxEKzAHSx4=","getTimezoneOffset","GUhoT1whbHk=","random","d2ViZHJpdmVy","AudioWorklet","ZRxUGyNwUSg=","DXR
8M0gZcwY=","pageXOffset","PkNPBHgtTTM=","dG90YWxKU0hlYXBTaXpl","console","Em8jaFQPJ1M=","ZRxUGyNyVy4=","KVgYX284GmQ=","1005nXrFLJ","Ah8zWERzMGg=","spawn","innerWidth","caches","Android","OSgIb3x
DDV0=","4999764IZcsTz","WebAssembly","query","Z1pWXSIxVGg=","U04iSRYkIHI=","brands","webView","getTime","ZH0VeiEQF0E=","T2JqZWN0LmdldE93blByb3BlcnR5RGVzY3JpcHRvcg==","sendBeacon","(any-hover:
none), (any-pointer: coarse)","30CuyZEq","[object
MimeTypeArray]","pdfViewerEnabled","ZRxUGyNyWio=","b1JeVSo/XGM=","Rlt3HAMxdy4=","Ah8zWEd0MWo=","platform","addEventListener","log","SBl5Xg1wf2w=","eEkJDj4lBzw=","ancestorOrigins","emit","default
View","setInterval","S3Y6MQ4bNQs=","DXR8M0sfcgg=","cHEBdjUdAEM=","Q34yOQUeNA8=","WGlpbh4HZ1g=","UBFhVhZ8ZWI=","W0YqQR0mKHA=","atob","Rlt3HAA7cy8=","isSecureContext","Ix4SGWZ0ECo=","Sld7EAw5fiA="
,"FwpmDVFhZTk=","KDlZPm1XWA0=","pageYOffset","rtt","prototype","MatchesSelector","TTQ8cwhYMkI=","STg4fw9TPUo=","aGFyZHdhcmVDb25jdXJyZW5jeQ==","dispatchEvent","format","GmcrYF8OJFA=","26712JKXvgr
","cy5CaTVDQ10=","Y15SWSYyVm0=","getPrototypeOf","register","matchMedia","RTw0ewBSNEg=","downlink","pixelDepth","performance","PkNPBHgsQTc=","architecture","getComputedStyle","notify","AudioWork
letNode","CXh4P08Ydgw=","LxIeFWp5GyI=","EXBgN1QabgM=","onLine","
Mobile/","GUhoT1wga3k=","getBoundingClientRect","userAgent","QSAwZwRJNlc=","WGlpbh4JZls=","ZH0VeiEVG0g=","getOwnPropertyNames","Z1pWXSIyV2Y=","YQBQByRrUzM=","dXNlZEpTSGVhcFNpemU=","geolocation",
"UTAgdxddJU0=","saveData","SBl5Xg55e2w=","AWBwJ0QNch0=","RequestAnimationFrame","enabledPlugin","toString","product","Ui9jKBdEZB0=","
Safari/","deviceMemory","HmMvZFsPKV4=","indexOf","ActiveXObject","keys","appCodeName","Performance","fEUNAjkrDDI=","Y3lwcmVzc1NlbmRUb1NlcnZlcg==","standalone","[object
Geolocation]","null","Chc7UE97P2M=","Bhs3XEN2MG4=","appVersion","value","BF01GkIzNSs=","AEExBkUvNjc=","connection","runtime","domAutomation","WGlpbh0Bal8=","innerHeight","referrer","DFU9Ekk5OiE=
","doNotTrack","MDFBNnVdRQQ=","EXBgN1ccZgw=","ZH0VeiIWFEk=","html","bHUdciobG0k=","map","navigation","setTimeout","PSQMY3hNC1k=","PointerEvent","domAutomationController","LnNfdGsYXEY=","productS
ub","ZjsXPCBWFwo=","maxTouchPoints","ICFRJmZBVhY=","Z1pWXSIyU2g=","documentMode","eWhILz8DSRU=","RunPerfTest","XGVtYhkIaVQ=","sort","NktHDHAqSTo=","
(pointer:fine)","UTAgdxdRL0Q=","cssFromResourceApi","language","WGlpbh0Db1s=","__nightmare","ZRxUGyBwWi8=","X0IuRRkiIHE=","RTw0ewNcMU4=","forEach","VitnLBNHZxo=","true","EventSource","openDataba
se","WQgoDx9mKz0=","platformVersion","RTw0ewNSOkg=","ondeviceready","Ui9jKBRAbBs=","DXR8M0gdfwE=","KndbcG8bWUs=","N2oGLXEEBxc=","XDomainRequest","fmget_targets","167cbSTsV","DFU9Eko6Mig=","Ix4SG
WVwECI=","scrollX","undefined","push"];return(Gh=function(){return t})()}function Dh(e){var
n=139,r=438,u=359,s=502,l=527,v=245,p=487,m=263,y=510,g=395,E=492,b=492,T=252,S=492,I=252,w=207,O=237,R=237,A=Rh;try{e[A(n)]=function(){var
e=229,n=224,r=229,a=229,o=201,i=200,u=205,s=229,l=_f,h="";if(!Zf)return h;for(var d=0,v=0;v<jf[l(e)];v++)try{d+=(Zf[jf[v]][l(n)]+"")[l(e)]}catch(t){}h+=d+Df;try{Zf[$f][ah](0)}catch(t){h+=(t+"")
[l(r)]+Df}try{Zf[$f][ah]()}catch(t){h+=(t+"")[l(a)]+Df}if(t(c[l(o)])===f&&0===c[l(o)][l(i)](l(u)))try{Zf[th][rh]()}catch(t){h+=(t+"")[l(s)]+Df}try{Zf[$f][eh][nh]()}catch(t){h+=(t+"")
[l(e)]}return h}(),e[A(r)]=function(){var t=229,e=229,n=_f,r=a[qf],i=r?(r+"")[n(t)]:0;return i+=Lf&&Lf[Kf]?(Lf[Kf]+"")[n(t)]:0,i+(o&&o[Jf]?(o[Jf]+"")[n(e)]:0)}
(),e[A(u)]=e[A(s)]=!!a[A(l)],e[A(v)]=e[A(p)]=i[zf]+"",e[A(m)]=e[A(y)]=zf in i?1:0,e[A(g)]=a[A(E)]&&a[A(b)][A(T)]&&a[A(S)][A(I)].id||"",e[A(w)]=t(a[A(b)])===d&&t(Object[A(O)])===h?Object[A(R)]
(a[A(b)]):[]}catch(t){}}function Lh(e){var n=223,r=281,i=517,u=357,s=452,f=256,h=187,d=469,v=465,p=381,m=482,y=Rh;oe(e,y(286),(function(){return vo()?1:0}),2),oe(e,y(n),(function(){var
e=y;return history&&t(history[e(m)])===l&&history[e(m)]||-1}),-1),e[y(r)]=ur(),e[y(i)]=ku,e[y(u)]=function(){var t={y:164,L:482,E:244,Q:311},e=Rh,n=[];try{var r=c[e(t.y)];if(c[e(t.y)])for(var
a=0;a<r[e(t.L)];a++)r[a]&&r[a]!==e(t.E)&&n[e(t.Q)](r[a])}catch(t){}return n}(),e[y(s)]=o[y(f)]?encodeURIComponent(o[y(f)]):"",e[y(h)]=a[y(d)](y(v))||!!a[y(v)],Ha&&(e[y(p)]=function(){var t=
{y:376},e=Rh;try{return null!==o[e(t.y)](0,0)}catch(t){return!0}}())}function Zh(t){}function Yh(e){var
```

Figure 3.3: Obfuscated script from the company Human.

Most of the behavioural biometrics scripts were either minified or obfuscated. We could not use a deobfuscator on the scripts since that would change the behaviour of the scripts and may remove the places where the taint was being lost. We had to keep the scripts as they were to find out exactly what set of conditions led to the loss of tainted data in order to fix FoxHound. Since we only had these scripts to work with, we were left having to decipher these complicated scripts. By doing that, we were able to gather some useful information on obfuscated scripts. To better understand the scope of this work, Figure 3.3 shows an example of part of an obfuscated script. We were able to extract meaning and flows from these types of scripts. These scripts tend to encode the data using some kind of encoding mechanism. They also have many functions that have different behaviours based on the input that is given. Names of common keywords are usually stored in an array and the array with the given index is used to call that keyword instead of the keyword itself making it harder to understand what the function does. Obfuscated scripts are also more likely to use data structures that are not dynamic like typed arrays. They also tend to do a lot of conversions to throw off readers. Our work with these obfuscated scripts helped us find a lot of the bugs in FoxHound. By fixing these bugs, we believe we made FoxHound more effective at tracking taint flows within obfuscated scripts. We are excited to see what other researchers can achieve with this upgraded version of FoxHound.

We also saw common patterns among the behavioural biometrics scripts that could help identify behavioural biometrics scripts faster. The first one is periodic network requests. Some of the scripts had periods that were based on time like every five seconds, while others used batching, which consists of sending over the data only when you reach a threshold. Periodically sending behavioural biometrics data over the network can potentially be used for continuous authentication. Another common pattern is data aggregation. Almost all of the scripts would take the behavioural biometrics data and aggregate it together in some sort of data structure. The last pattern we noticed is that the data was always encrypted by the script before being sent over the network.

3.5.3 Limitations

While the modifications significantly improved FoxHound's effectiveness, certain limitations remain:

- Conditional Branch Taint Propagation:
  - FoxHound currently does not propagate taint through conditional branches within decision trees. This limitation, observed in scripts like Forter, stems from the risk of over-tainting and losing precision in taint tracking.

- Performance Overhead:
  - Some enhancements, such as disabling inline caches to ensure proper taint tracking, introduce performance trade-offs by forcing the code to execute in slower, taint-aware paths.
- Dynamic and Obfuscated Code:
  - While FoxHound performs well on many obfuscated scripts, certain highly dynamic constructs (e.g., scripts that generate code at runtime based on conditions that are hard to get) remain challenging to analyze.
- Limited Testing Scenarios:
  - Our evaluation relied on the scripts we identified during testing. There may be edge cases or new techniques in emerging behavioural biometrics scripts that FoxHound has not yet addressed.

Despite these challenges, FoxHound's ability to detect taint flows has improved dramatically, and the tool remains adaptable for addressing future issues in dynamic taint analysis.

3.6 Conclusion

We were able to identify an approach for finding behavioural biometrics scripts using dynamic taint analysis. We then looked at all the different types of dynamic taint analysis and concluded that FoxHound is the best tool for our needs. We then found 19 behavioural biometrics scripts from which we were able to narrow it down to 15 that were actively being used by websites. We tested the original version of FoxHound and found that only three out of the 15 taint flows were being caught representing a 20% effectiveness. We then did nine separate changes to Foxhound, which helped improve the effectiveness to 14.5 scripts found bringing our effectiveness to 97%.

The enhancements made to FoxHound mark a substantial leap in its ability to analyze complex behavioural biometrics scripts. By implementing boolean tainting, fixing taint propagation in special cases, and ensuring robust inter-process communication tracking, we increased its effectiveness from 20% to 97% for behavioural biometrics scripts. These changes allow FoxHound to address scenarios that were previously untraceable, significantly improving its utility in real-world applications The progress achieved highlights the value of iterative improvement in dynamic taint analysis. FoxHound now stands as a more capable and adaptable tool. It can now be used for tracking behavioural biometrics scripts.

Chapter 4

Checkout Crawler

This chapter will discuss a novel checkout crawler that was created to go from the main page of an e-commerce website to the checkout page. This crawler will later be used to find behavioural biometrics scripts.

4.1 Introduction

An e-commerce website can be defined as an online store where customers can visit, look at products and purchase them [4]. The users that purchase these items will need to provide a form of payment. Some possible forms of payment are credit cards, debit cards or gift cards. A fraudulent transaction is when a form of payment is used without the consent of the user to whom the form of payment belongs. There are several ways to prevent fraudulent transactions and one of the ways is to ensure the person completing a purchase is who they say they are.

However, determining if an e-commerce website employs behavioural biometrics is not straightforward. To address this, we developed a novel checkout crawler designed to simulate typical user interactions on a website. Our crawler navigates from the main page to a product, adds the item to the cart, and attempts to reach the checkout page. Our crawler has an accuracy of 78% and a precision of 98% on Shopify pages, that is, our crawler reaches 78% of existing checkout pages and when it reports reaching a checkout page, it is correct 98% of the time. This crawling can help identify scripts aimed at behavioural verification, providing valuable insights for researchers and developers.

4.2 E-Commerce Website Flow

Understanding the flow of a traditional e-commerce website is essential for building an effective checkout crawler. To gain this insight, we conducted a detailed analysis of the top 100 Shopify stores obtained by taking the first 100 entries from this list [2]. Shopify was chosen as our focus platform because it is the most widely used for creating e-commerce websites, offering a diverse range of site structures and purchasing flows.

By examining these 100 websites, we aimed to identify common patterns in the user journey from browsing to checkout. A general flow for e-commerce websites can be seen in Figure 4.1. The flow we presented is meant as a generalization of all possibilities in the e-commerce flow. Some websites do not have some connections and other websites do not have all of the pages mentioned in the flow. The only two pages that are guaranteed in an e-commerce website are the main page and the checkout page and the only requirement is that there is at least one item in the cart. This section will outline the primary pages encountered during this process and describe their characteristics.

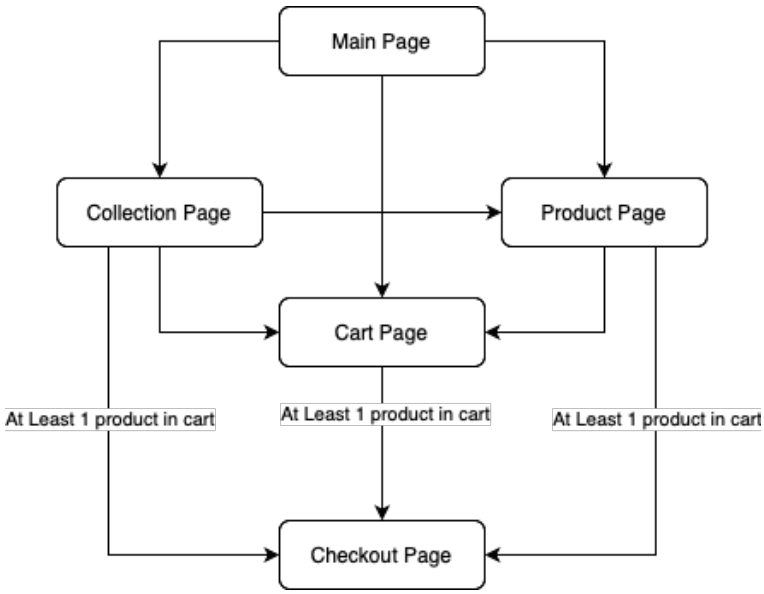


Figure 4.1: E-Commerce checkout flow

4.2.1 Main Page

The main page is the initial landing point of an e-commerce website, often showcasing featured products, curated collections, and promotional content. Some websites offer a "quick buy" feature, allowing customers to make purchases directly from this page. Other websites offer the ability to add items to a cart on the main page. However, in many cases, the main page primarily displays collections, which are groups of related products and products that cannot be bought on this page. Examples of main pages can be found in Figure 4.2.

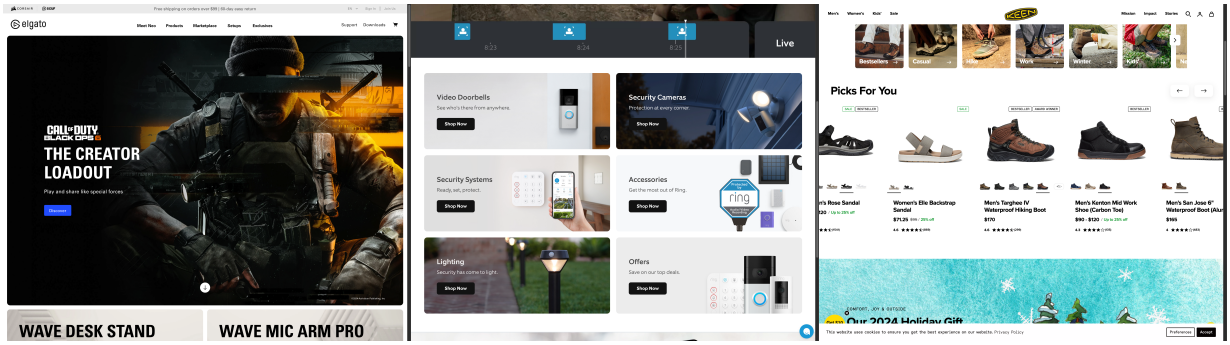


Figure 4.2: Main pages examples. From left to right, Elgato's website has promotional content. Ring's website shows different collections. Keen's website has products that can be bought directly.

4.2.2 Collection page

A collection page can be seen in Figure 4.3. It organizes products into specific categories, such as a summer collection featuring items suitable for warm weather. These pages often include filters that help users narrow their search by criteria such as price, size, and colour.

4.2.3 Product Page

A typical product page can be seen in Figure 4.4. It provides detailed information about a specific item, including a description, price, and additional specifications. Users may also see related products displayed at the bottom of the page. There is also the option to choose the size of the product that you want to buy if applicable. This is dependent on the product

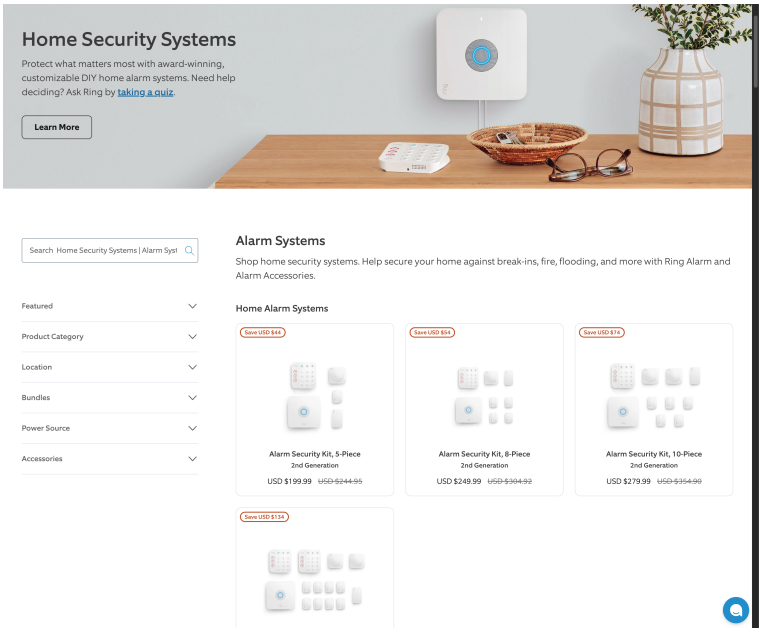


Figure 4.3: Ring's alarms collection page with filtering abilities on the left.

type. For items such as shoes or clothing, users can select a size from options such as 6-12 for North American shoe sizes or XS-XL for shirts. The "Add to Cart" button, sometimes labelled as "Bag," "Basket," or "Tote," allows the user to prepare their purchase. In some cases, selecting a size or other attributes is mandatory before adding the item to the cart.

4.2.4 Cart Page

Once products are added to the cart, users can go to the cart page to look at all the items they have chosen to get, which can be seen in Figure 4.5. The cart page lists all selected items, displaying individual prices and the total cost. Users can often adjust the quantity or remove items as needed. Some websites do not have a cart page, but rather a banner that appears on the current page with the item in the cart once a product is added to the cart. No matter the type of cart a website has, a "Checkout" button will be found that directs users to begin the payment process.

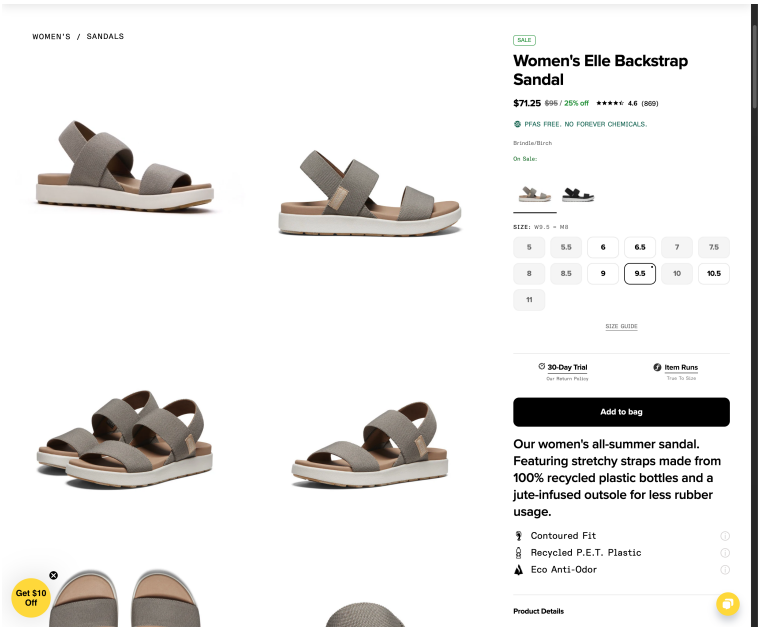


Figure 4.4: A product page from Keen's with a product description, the ability to select size and to add the product to the cart.

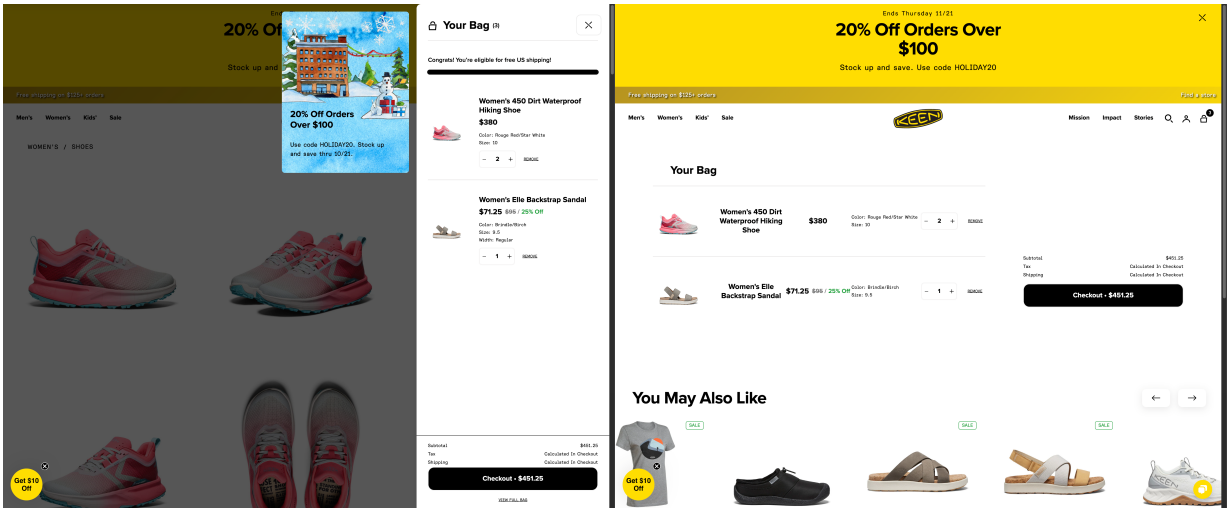


Figure 4.5: Keen's has both a cart page and a cart banner. Both have the checkout keyword.

4.2.5 Checkout Page

Once the checkout button is pressed, the user is usually directed to a checkout page, which can be seen in Figure 4.6. The checkout page is where users finalize their purchase, often with options for a quick checkout, account sign-in, or guest checkout. Users provide shipping and payment details, which may be collected over several steps. For our purposes, successfully reaching this page signifies a successful crawl.

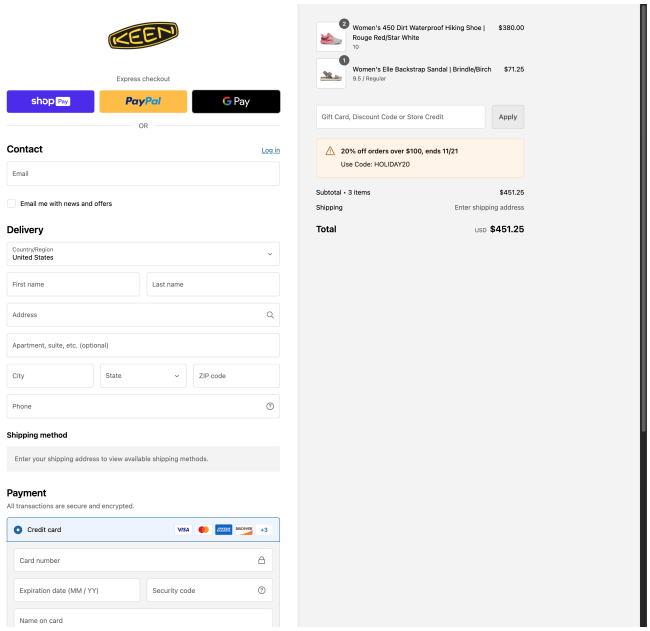


Figure 4.6: Keen's checkout page.

4.3 Crawler Design

The design of our checkout crawler is guided by the insights gained from analyzing e-commerce website flows. The primary goal was to create a flexible yet effective crawler that could handle a variety of e-commerce structures without being overly tailored to specific cases. Striking a balance between generality and performance was crucial to ensure broad applicability across different platforms.

4.3.1 Infrastructure

The checkout crawler is built on a robust infrastructure designed for easy extensibility and customization. We selected Klein et al.'s tool [33] as the foundation due to its modular design and advanced features. We then customized it by adding our own checkout module, which contains the logic used to go from the main page to the checkout page. We chose Klein et al.'s tool [33] for the following reasons:

- It can track visited websites and their status in an easy-to-use database.
- It has a customizable retry mechanism for failed requests.
- Easy to extend functionality with custom modules by abstracting how the browsing and error handling works.
- Switch seamlessly between browsers, including the FoxHound browser.
- Use Playwright, which is the only automation tool supported by FoxHound.
- Run multiple crawler instances in parallel, improving efficiency and scalability.

4.3.2 High Level Design

For each website, we will first gather 31 web pages that we need to crawl and for each, we will run the modules we have. Since all e-commerce websites are different, no one flow can fit them. For this reason, we have created four modules that each run a slightly different setup to ensure we have the highest possible coverage. Each module is composed of components, which are the size selection component, add to cart component, go to cart component and go to checkout component. A more detailed description of each component can be found later on. The high-level design can be seen in Figure 4.7.

4.3.3 Crawling Modules

The first module is called the Simple module. This module will go to the page currently being crawled. The module will call the add to cart component. If it detects that one or more items were added to the cart, it will call the go to checkout component.

The second module is called Cart. The difference between the Simple and Cart module is that in the Cart module, when one or more items are added to the cart, the go to cart

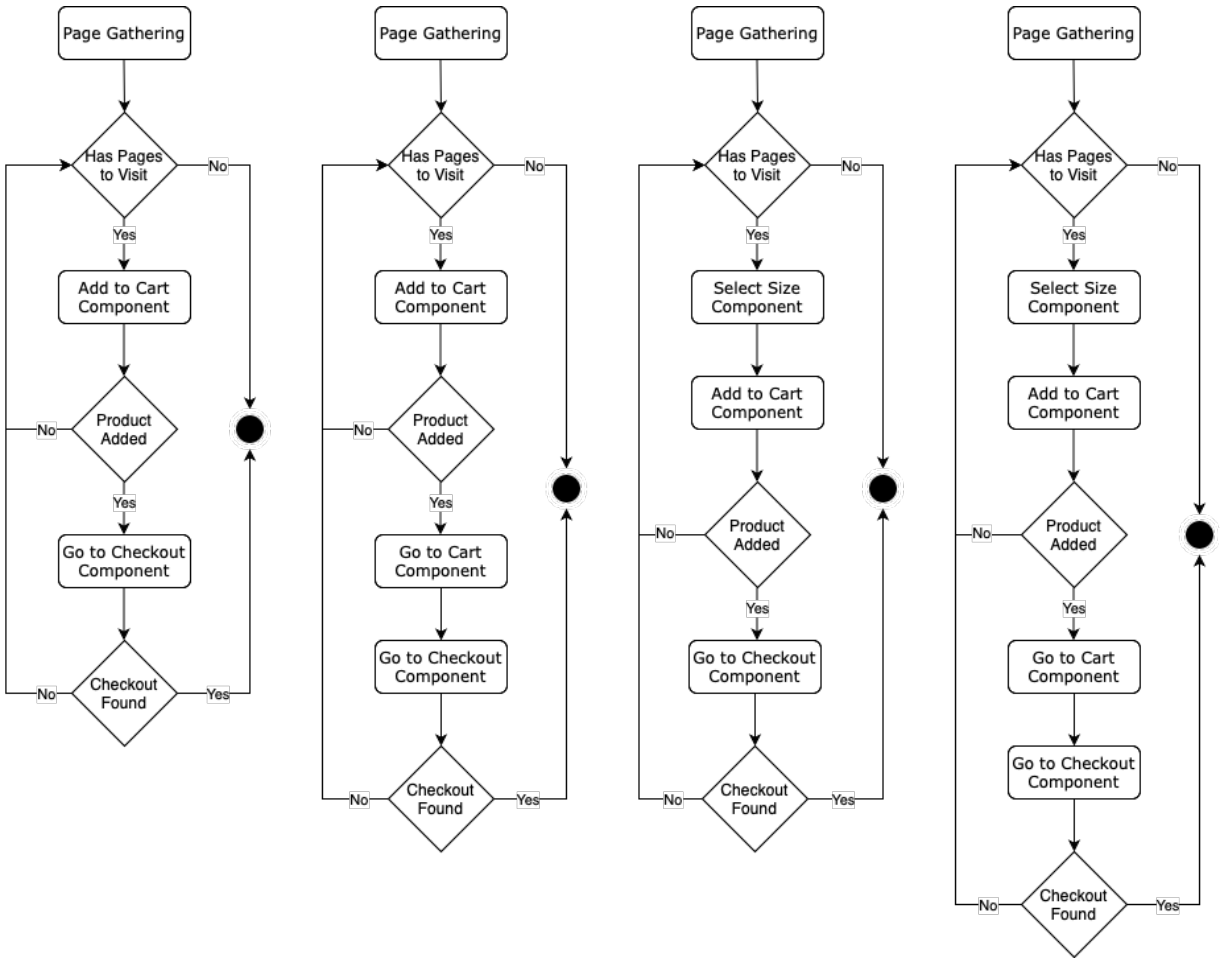


Figure 4.7: Crawling Modules. From left to right: Simple module, cart module, size module and cart and size module

component is called instead of the checkout component. After the go to cart module is finished, the checkout component is called.

The third module is called Size. This module is similar to the Simple module but before calling the add to cart component, it will call the size selection component.

The last module is called Size and Cart. It combines the changes from the Cart module and Size module into one. So the flow of this module is it starts by calling the size selection component, followed by the add to cart component. If items were added to the cart, the go to cart component is called followed by the checkout component.

One optimization that we did was to stop the crawl for a website when a checkout page is reached. So instead of crawling all 31 pages that we gathered, we only crawl the web pages until we have reached a checkout page and then we stop the crawl for that website. Another improvement we made was to use the extra stealth plugin with our crawler [11]. This allowed us to avoid many bot detection scripts and allowed us to visit more websites. We made our crawler [10] open source and free to use.

4.3.4 Page Gathering

The first part of the checkout crawler is the page gathering. From the main page, we have decided to take ten random links that are found and store them in a database. For each one of the links found we then pick a random link from that webpage and add it to the database. We do the same process again for the new found links. At the end, we will have 31 web pages to look at.

We decided to go down this path since the main page usually does not have many products on it. We take ten random links in the hopes that one of them is a collection page or a product page. From our observations, this seems to be the case most of the time. For each new web page, we only find and record one extra link. We do this since we are trying to find product pages. If we are on a collection page, most of the links will be product page links. So by randomly picking a link we are likely to find the product page. We also only look at one link for efficiency. The more web pages we look at, the longer the crawl will take. We found this to be the best compromise between speed and efficiency. The last choice we made was to go to a depth of three for the web pages (1 main page and 10 pages at each other level for the 31 total we have). This is done because some websites have product pages deep in the navigation chain. Having a higher depth allows us to be more likely to find these products.

4.3.5 Size Selection Component

The first component is the size selection. In this part, the crawler will look to select the size of a product. The sizes we are looking for are the following three: "l", "large" and "10". These sizes were chosen based on the most common sizes that have been seen in the top 100 Shopify websites. This component does not support more complicated size selection that involves colours or using dropdown menus. It is also important not to look for too many sizes, since that can lead to accidentally leaving the webpage we are currently on.

4.3.6 Add to Cart Component

The second component is the add to cart component. In this component, the crawler will look for the following keywords: "add to cart", "add to bag", "add to basket", "add to tote" and "add". These keywords were chosen based on our observations from the top 100 Shopify websites. The crawler will take all buttons that contain these keywords and click on up to ten of these buttons. After the "add to cart" button is clicked, some websites redirect users to a product page, while others add the item to the cart without leaving the current page. If we detect that the webpage is no longer the same, we assume that we are now on the correct product page and we re-run this component. We only do this with a depth of two. We also make sure that if a button from the keywords is pressed properly, we stop pressing buttons that contain one of the keywords. This is done to speed up the time taken per crawl.

4.3.7 Go to Cart Component

The third component we have is the go to cart component. This component is in charge of going to the cart. It will look through all the links on the page and filter them based on which ones contain the word cart in it. It will then sort the links based on length and visit the shortest link. This approach has been taken since almost all webpages from the top 100 Shopify list have a cart page with the link in the format of the website name followed by the word cart.

4.3.8 Go to Checkout Component

The fourth and last component is the go to checkout component. From the most common e-commerce flow, the typical keywords used for checkout are the following: "Checkout", "Check out", "Proceed", "Place Order" and "Complete Purchase". For this part, we look for buttons or text in the HTML that contain these keywords.

However, some descriptions not related to the checkout page might have some checkout keywords in them. An example would be the following text: "Taxes will be calculated at checkout". If this text is clicked, it will bring the user to a page that shows how taxes are calculated resulting in not being able to reach the checkout page. To solve this issue, we sort the HTML tags containing keywords based on length to ensure we first click on the shorter texts.

For some websites, clicking a checkout button will bring the user to the start of the checkout flow while on other websites it will bring the user to the cart where there is another checkout button that will go to the start of the flow. For this reason, we re-run the checkout component once we have arrived at a different page to ensure we did not miss the flow where we are re-directed to the cart page. This component will also take a screenshot if a new page is reached during this flow, which we will later use to test the accuracy of the model.

4.4 Evaluation

Our checkout crawler represents a pioneering effort in automating the journey from product selection to the checkout page on e-commerce websites. Since no existing tools or datasets could be used for direct comparison, we devised our evaluation methodology. This section outlines how we created a custom dataset and measured the crawler's performance.

4.4.1 Datasets

To evaluate our crawler, we first examined e-commerce flows using a dataset of the top 1000 Shopify websites [2]. This allowed us to measure the crawler's performance in a familiar, well-structured environment. However, to test the crawler's generalizability, we needed a more diverse dataset beyond Shopify-based sites.

Existing datasets of top e-commerce websites often included non-commercial sites, such as news platforms, blogs, product information websites with no ability to purchase items

and many more types. To address this, we manually curated a set of 118 true e-commerce websites from the top 200 Canadian e-commerce sites [13]. The list can be found in Appendix C. This refined dataset, alongside our crawler [10], provides a robust foundation for our evaluation.

4.4.2 Experiment

We conducted our experiments on a server running Ubuntu 22.04.5 LTS (Jammy Jellyfish) with an Intel Xeon E7-8870 CPU (10 cores, 20 threads) and 256 GB of RAM. To maximize efficiency, we ran 15 parallel instances of our crawler. The crawler status and results were stored in a self-hosted MySQL database.

For evaluation, the crawler captured screenshots upon reaching the checkout flow. We defined:

- True Positive (TP): A screenshot correctly showing the start of the checkout flow.
- True Negative (TN): No screenshot, with no checkout page present.
- False Positive (FP): Screenshots taken, but none showing the checkout flow.
- False Negative (FN): No screenshot, despite a checkout page being present.
- Accuracy: The proportion of correctly predicted instances (both true positives and true negatives) among the total number of instances. It measures how often the crawler makes a correct prediction.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- Precision: The proportion of correctly predicted checkout pages (true positives) out of all pages predicted as checkout pages (true positives and false positives). It measures the crawler's ability to correctly identify a page as a checkout page.

$$\text{Precision} = \frac{TP}{TP + FP}$$

- Recall: The proportion of correctly identified checkout pages (true positives) out of all actual checkout pages (true positives and false negatives). It measures the crawler's ability to detect all existing checkout pages.

$$\text{Recall} = \frac{TP}{TP + FN}$$

Metric	Top 1000 Shopify	Top 118 Random
True Positives	699	64
False Positives	14	4
True Negatives	59	0
False Negatives	199	50
Failed to Load	29	0
Accuracy	0.78	0.54
Precision	0.98	0.94
Recall	0.78	0.56
F1 score	0.87	0.70

Table 4.1: Checkout Crawler Results

- F1 Score: The harmonic mean of precision and recall. It provides a single metric to evaluate the predictive performance of the crawler, especially useful when there is an uneven class distribution.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

4.4.3 Results

We tested our crawler on two datasets: the top 1000 Shopify websites and our curated set of 118 Canadian e-commerce sites. Table 4.1 summarizes the outcomes of these experiments, highlighting the crawler's performance in different contexts.

4.5 Discussion

In this section, we will expand on the findings from Table 4.1.

4.5.1 Top 1000 Shopify

Accuracy

From Table 4.1 we can see that we got 78% accuracy, which represents how well our crawler can go to checkout pages. This is a strong result, highlighting the commonalities across

e-commerce pages, which is probably due to the usability needs. Each website wants to be familiar to new users while at the same time differentiating itself from others. They often differentiate themselves through design choices such as colour schemes, typography, layout spacing, and specific keyword usage, creating a unique brand experience. But to make sure the users know how to purchase items, the purchasing flow is always quite similar. A user needs to go to a product page, add the product to the cart and then go to the checkout pages. We believe that we were able to get high accuracy because most websites want to have a familiar flow.

Precision

The precision of the checkout crawler is 98%. This shows that when the checkout crawler produces screenshots, it is fairly certain that a checkout page was reached. We believe this is due to the go-to checkout keywords. These checkout keywords are usually reserved to complete a purchase and that makes e-commerce companies less likely to use these keywords for other purposes. So whenever the crawler finds a checkout button and clicks it, it is likely that it will bring the user to the start of the checkout process.

Loading Failed

While manually classifying the crawler results, we have noticed some interesting patterns. The first one is that even if Shopify websites are popular, that does not mean that they are always up and running. Sometimes, the websites are out for maintenance, the website, URL has been bought by another company or the URL is no longer in use. This is shown by the 29 pages that failed to load when manually inspecting the top 1000 websites.

Non E-Commerce Pages

Another interesting finding we came across is that some companies that built websites using Shopify do not use the websites for e-commerce purposes. For example, MSNBC uses Shopify for their website but the company is in the business of providing news, not e-commerce. They do not have a checkout page on their website and our crawler correspondingly was unable to find one. Other companies have product pages, but you cannot buy from the website. They may provide a link to an Amazon store, indicate where the product can be found in nearby stores, or simply advertise the product without offering a direct purchase option. Another use case we have seen that is not e-commerce is for

university websites or charity websites. The total amount of websites that did not have a checkout page was 59 out of 971 active websites, which means that 6% of Shopify websites are not e-commerce. We chose to retain the original Shopify website list for future analysis and did not filter out websites that lack a checkout page. The takeaway is that Shopify websites are mostly e-commerce websites but certain companies use Shopify to build their websites without using the e-commerce platform Shopify offers.

Checkout Flows not found

False negatives are websites that have a checkout page but our crawler did not find them. We had 199 false negatives and we noticed some patterns in the websites we were not able to get to the checkout page. One reason for the crawler failing to find checkout pages was bot detection. Some pages have bot detection scripts, which can detect that the crawler is not a human accessing the webpage and it stops loading the page. This can result in timeouts or prevent the e-commerce flow from being executed successfully.

Another reason for the false negatives is hard to find products that require a high amount of navigation to get to them. Some companies do not primarily sell items on their websites. They have another purpose for their websites and the products they sell are on a hard-to-reach page. For example, a blogging website may host thousands of blog posts and related pages. Although the site also sells branded t-shirts, accessing the purchase page is challenging because it is deeply nested within multiple sub-pages. Since our crawler primarily navigates through random pages using a few heuristics, it is highly likely to overlook these deeply embedded pages.

Some pages use keywords that we have not anticipated for the different parts of the checkout process. For example, there is an e-commerce store selling baby products that uses “place in the nursery” for the add to cart flow. This also happens in the size selection flow. Our heuristics are meant to capture the most amount of flow without taking too much time and producing false positives. If we add too many keywords, the number of false positives would increase and our precision would decrease.

Complicated size selection is another aspect that caused issues for our crawler. Not all of the websites allow to easily click on the size to select it. Some websites require users to select multiple product characteristics before allowing the item to be added to the cart. Other websites have drop-down menus that only offer the option of choosing the size after opening the menu. There are even websites where you have to manually enter aspects about the product like the quantity to buy.

Some websites are not English-native websites. In our dataset, we encountered Japanese, French and Spanish websites, which our crawler failed to get to the checkout page. The keywords that our crawler looks for cannot be found on websites built in a different language.

The last issue we faced was the opening of new tabs or new windows. Some websites have a different URL for shopping pages compared to their normal websites. Usually, the store URL is composed of "shop." followed by the URL of the website. The websites would open a new tab or page with this new URL. This caused the crawler to think it left the website and stop crawling it. The same thing also happens with checkout pages. Some websites open a new secure window to complete the checkout process and the crawler cannot detect the new window that was opened.

4.5.2 Top 118 Random

Accuracy

The accuracy of the checkout crawler for random e-commerce websites is 54%, which is lower than the accuracy of Shopify web pages. We believe this is because the checkout crawler was built with heuristics gathered from a Shopify dataset. The accuracy was expected to drop but it is still high in our opinion. This means that our checkout crawler also works on non-Shopify pages and that our heuristics translate well to other e-commerce websites. The drop in accuracy tells us that the purchasing flow varies more on non-Shopify websites.

Precision

The precision is 94%, which is high and did not change much from the Shopify experiment. This result tells us that non Shopify e-commerce websites also have checkout keywords that are reserved for the payment flow. This is a key insight that is useful for building more robust and accurate checkout crawlers in the future.

Checkout Flows not found

We do not have any true negatives or websites that failed to load. This is because we manually created this dataset as there was no previous dataset of top e-commerce websites

that allow for the purchase of items. Since we created it, we made sure to not add websites that fail to load or websites that are not e-commerce.

The reasons our crawler failed to find checkout pages are similar to the issues it encountered on Shopify websites. We noticed that size selection for non-Shopify websites is a lot more complex than for Shopify websites and we believe that our heuristics for selecting sizes is the main reason why our crawler did not perform as well on this dataset.

4.6 Future Work

Our checkout crawler is a pioneering effort, but there is significant potential for further enhancement. We propose several directions for future research and development based on our findings:

- **Improving the Size Selection Component:** Our current heuristic approach struggles with websites that deviate from expected size-selection conventions. Expanding the set of keywords led to more false positives, reducing precision. Additionally, language barriers pose a challenge, as keywords are often language-specific. Future work could explore adaptive algorithms or context-aware models to better handle size selection across diverse websites and languages.
- **Incorporating Machine Learning:** Many e-commerce sites vary significantly in structure, making it difficult for a heuristic-based crawler to adapt. Machine learning models, such as those utilizing natural language processing (NLP) or computer vision, could be trained to understand page context and dynamically determine the next steps in the checkout flow. This approach could make the crawler more flexible and robust across different platforms.
- **Extending the Checkout Flow:** Currently, our crawler stops at the beginning of the checkout process. To gain a deeper understanding of user verification mechanisms, future iterations should explore multiple payment methods, such as Shop Pay, Apple Pay, and PayPal. Different scripts may be triggered depending on the chosen payment method, offering richer data for analysis.

4.7 Conclusion

Our research introduces the first heuristic-based checkout crawler capable of navigating e-commerce websites to reach the checkout page. This novel tool addresses a gap in the automated analysis of e-commerce flows and contributes a valuable method to get to checkout pages, which are considered sensitive pages.

Our crawler demonstrated impressive performance, achieving 78% accuracy and 98% precision on Shopify websites, thanks to the predictable nature of Shopify's purchasing flows. When tested on our custom dataset of 118 diverse e-commerce websites, the crawler achieved 54% accuracy and 94% precision. These results highlight the strengths of our approach while also pointing to areas where adaptability is needed.

In addition to presenting a new automated tool, we curated a comprehensive dataset of e-commerce websites that can serve as a resource for future research. Moving forward, our work lays the foundation for more sophisticated crawlers, potentially incorporating machine learning to enhance accuracy and expand coverage.

Chapter 5

Machine Learning Pipeline

This chapter will discuss the machine learning pipeline we built to classify behavioural biometrics scripts.

5.1 Introduction

Dynamic taint analysis generates taint reports by tracking the flow of data from defined sources to corresponding sinks. Our analysis revealed that behavioural biometrics scripts can be effectively detected using dynamic taint analysis, where user-triggered events serve as sources and network transmission mechanisms act as sinks. However, behavioural biometrics scripts are not the only category of scripts exhibiting these data flow patterns.

5.1.1 Script Variants in Web Analytics

Modern web analytics involve a variety of scripts that collect user interaction data. In addition to behavioural biometrics scripts, two prominent categories that generate similar taint flows are session replay scripts and heatmap scripts.

Session Replay Scripts

Session replay scripts are designed to reconstruct user sessions, enabling companies to analyze detailed user interactions. These scripts capture events such as mouse movements,

scrolling, and clicks, then transmit the collected data to backend servers via network requests. This process facilitates the identification of usability issues and supports improvements in website layout and user experience.

The data flow pattern of session replay scripts closely mirrors that of behavioural biometrics scripts because both rely on user-generated events and network-based data transmission. Consequently, dynamic taint analysis is capable of detecting session replay scripts, which underscores the challenge of distinguishing between different script types solely on the basis of taint reports.

Heatmap Scripts

Heatmap scripts offer another perspective on user engagement by visualizing interaction data. They monitor click events and other interactions, aggregating this information to create heatmaps that highlight areas of high engagement. These visualizations enable companies to optimize website layouts and enhance overall usability.

Much like session replay scripts, heatmap scripts rely on event-based data collection and network transmission. The resulting taint report patterns are similar to those produced by behavioural biometrics scripts, further complicating the task of script classification when relying exclusively on dynamic taint analysis.

5.1.2 The Role of Machine Learning in Script Classification

Manual classification of taint reports is infeasible due to the sheer volume of scripts and the subtle differences between behavioural biometrics scripts and session replay or heatmap scripts. Traditional heuristics-based approaches struggle to generalize across new, unseen tracking mechanisms. Machine learning provides a scalable solution by learning statistical patterns in taint flows, allowing for robust classification across diverse datasets.

To overcome these challenges, we leverage machine learning models to analyze taint reports and extract discriminative features. By employing advanced feature extraction techniques and classification algorithms, our approach enhances the accuracy of distinguishing behavioural biometrics scripts from other tracking scripts. The following sections detail the development of our machine-learning pipeline and evaluate its performance in script classification.

5.2 Building Ground Truth

The foundation of any machine learning classifier is a well-defined ground truth dataset. Since no publicly available ground truth exists for taint reports caused by behavioural biometrics scripts, we constructed our dataset. This process required three key components: (1) a web crawler capable of simulating user interactions, (2) a storage system for collecting and managing taint flow data, and (3) a curated list of websites known to use behavioural biometrics scripts.

5.2.1 Simulated User Module

Our crawler builds on the framework [33] initially discussed in the checkout crawler chapter. We extended this crawler by integrating a new module that simulates genuine user behaviour. This enhancement enables the crawler to trigger events that closely mimic those of a real user, thereby producing more realistic interaction data.

The crawler is configured to execute a predefined sequence of events consistently across all websites, ensuring uniformity during both the ground truth data collection and the large-scale analysis phases. The sequence of simulated events is as follows:

1. **Click Events:** Four click events are triggered at random locations on the webpage, with a 250ms delay between each click.
2. **Double Click Event:** A single double-click event is executed.
3. **Mouse Movement:** The cursor moves from one random location to another.
4. **Mouse Down and Mouse Up:** A mouse press-and-release action is performed.
5. **Keyboard Events:** Key presses, along with key-down and key-up actions, are simulated.
6. **Copy-Paste Action:** A copy-and-paste event is triggered.
7. **Scrolling:** The crawler simulates both upward and downward scrolling.
8. **Screen Resize and Orientation Change:** The screen is resized to a random dimension, followed by an orientation change.

9. **Touch Events:** A multi-touch interaction is simulated by triggering touch events at random positions on the page.

When executed within our taint-aware browser environment, this enhanced crawler generates taint reports each time a taint flow reaches a defined sink. Integrated event listeners capture and store these reports, thereby facilitating subsequent data analysis.

5.2.2 Storage

The taint reports generated by the crawler contain essential metadata, including the script responsible for the report, the sources and sinks involved, and detailed information on the taint propagation flow. Given that this data is naturally structured in JSON format, we selected MongoDB as our storage solution.

MongoDB is a document-oriented NoSQL database that excels at handling hierarchical and unstructured data. Its flexible schema design allows us to efficiently manage and query large volumes of taint reports, which is crucial during both the initial data collection phase and subsequent large-scale analyses. Furthermore, MongoDB's scalability and performance ensure that our storage system can accommodate the growing dataset as more websites and taint reports are incorporated into the study.

5.2.3 List of Websites

To compile a comprehensive dataset of behavioural biometrics scripts, we leveraged the curated list of websites identified in joint work with Aswad Tariq, which we discussed in Section 3.4.1. This list comprises 114 websites and represents a diverse array of companies known to provide behavioural biometrics services, which can be found in Appendix A. It is important to note that not all websites actively use these services, and it is possible that some scripts are not active or that the behavioural biometrics service is turned off.

We also added 20 websites to our crawl that are known to contain different web analytic scripts that use mouse and keyboard data. These 20 websites have been found by Zhang [54]. The websites were found by searching for different web analytic companies and finding their clients. The client's websites were then searched for the web analytic script. The full list of behavioural biometrics websites and web analytic websites can be found in Appendix A.

Each website on the list was systematically crawled using our custom-built crawler, which was designed to simulate realistic user interactions. By applying a consistent crawling methodology across all domains, we ensured that the taint data collected was uniform and reliable.

5.2.4 Ground Truth

After executing our crawler across the identified websites, we populated a MongoDB database with taint reports from various scripts. In total, taint data was collected from 126 distinct scripts, with some websites contributing multiple scripts while others produced none. To establish a reliable ground truth, we manually analyzed each script to determine whether it exhibited behavioural biometrics characteristics.

To classify the collected scripts as either behavioural biometrics or non-behavioural biometrics, we employed a systematic, multi-step approach:

1. **Reference to Known Behavioural Biometrics Scripts**: We began by comparing each script against a repository of known browser behavioural biometrics scripts developed in collaboration with Aswad Tariq described in Section 3.4.1. This initial comparison allowed us to quickly identify scripts with previously documented behavioural biometrics tracking.

2. **Examination of Script Metadata**: For scripts not identified in our initial repository, we scrutinized the script headers for any company identifiers or metadata. If a company name was present, we researched the organization's services to determine if they were associated with behavioural biometrics or related tracking technologies.

3. **Keyword Analysis**: In instances where no explicit company information was available, we conducted a keyword analysis within the script. Specific terms known to be associated with behavioural biometrics activities, such as `wupServerURL` (commonly found in BioCatch scripts) served as indicators of behavioural biometrics functionality. Additionally, we searched for general web analytics related keywords (e.g., analytics, marketing, tracking, fingerprint) to identify scripts likely focused on non-behavioural tracking or data collection.

4. **Analysis of Data Exfiltration Endpoints**: When the above methods did not yield definitive classification, we analyzed the endpoints to which the script transmitted data. By examining the destination URLs and conducting public scans, we aimed to

associate the data recipients with known entities or services. This helped infer the script's purpose based on the nature of the receiving organization.

Through this comprehensive methodology, we successfully classified all 126 scripts without any remaining undetermined.

While 126 scripts may seem limited, it provides a crucial first step toward automated classification. The dataset covers a diverse range of tracking scripts with different ways of tracking user's actions, allowing the model to learn general patterns. However, future work should expand the dataset using additional sources to improve generalization.

This investigation resulted in the following classification:

- 56 scripts were confirmed as behavioural biometrics scripts.
- 70 scripts were classified as non-behavioural biometrics scripts.

The set of scripts that were not behavioural biometrics consisted of heatmap, web analytics, session replays and tracking scripts. This annotated dataset now serves as the ground truth for training and evaluating our machine-learning model.

5.3 Feature Extraction

With the ground truth dataset established, the next step is to extract meaningful features from each script to train our machine learning model. These features are derived from a script's taint reports, which provide detailed information about data propagation—including sources, flows, and sinks. Since a taint report is generated each time tainted data reaches a sink, scripts that aggregate data before transmission produce fewer taint reports compared to those that transmit data directly.

5.3.1 Feature Engineering

To construct our feature set, we process each taint report as follows:

1. **Script Involvement:**
    - Extract all scripts that contributed to a given taint report.

- Store all taint reports associated with each script.

2. **Source and Sink Frequency:**
   - For each script, count the occurrences of source events present in its associated taint reports. This count reflects the volume of sensitive data (mouse, touch and keyboard events) that are being exfiltrated by the script.
   - Similarly, count the occurrences of sink events linked to the script. This count illustrates both the frequency and the mechanisms of data exfiltration (e.g., network request, socket communication, etc.).
3. **Taint Report Volume:**
   - Measure the total number of taint reports generated by each script. A higher number of taint reports suggests more frequent interactions with user-generated events.
4. **Taint Flow Characteristics:**
   - Compute the total number of taint flows per script, as each taint report may contain multiple flows. Scripts with a higher number of taint flows likely process and modify user interaction data extensively before reaching a sink.
   - Track the total number of nodes within the taint flows and sum them up. Each node represents an operation performed on tainted data, such as arithmetic calculations or string manipulations. Behavioural biometrics scripts often perform multiple transformations on user interaction data before sending it to a sink, distinguishing them from simpler tracking mechanisms.

5.3.2 Feature Overview

Table 5.1 summarizes the key features extracted from the taint reports:

These extracted features provide valuable insights into how scripts interact with tainted data, helping our model differentiate between behavioural biometrics scripts and non-behavioural scripts.

Feature	Description
Individual Source Count	Total number of source events (e.g., user-triggered events) extracted from the taint reports for a given script. Full list of sources can be found in Appendix B.
Individual Sink Count	Total number of sink events (e.g., network transmission endpoints) recorded in the taint reports for a given script. Full list of sinks can be found in Appendix B.
Taint Report Volume	The overall number of taint reports generated by a script, reflecting the frequency of taint flows reaching a sink.
Taint Flow Count	The total number of taint flows aggregated across all taint reports for a script.
Node Count	The sum of all nodes across taint flows, indicating the complexity and the number of operations performed on tainted data.

Table 5.1: Extracted Features from Taint Reports

5.4 Evaluation

We developed a streamlined machine learning pipeline that leverages both the ground truth data we created and the features extracted from the processed scripts to classify behavioural biometrics scripts. The pipeline integrates the ground truth labels for each script with the corresponding feature set, which is then used to train a range of machine-learning models. The best-performing model is selected based on its performance metrics. To enhance the model's ability to correctly identify behavioural biometrics scripts, we implement an oversampling strategy that doubles each script labelled as behavioural biometrics in the set.

To pick our machine learning model, we need to evaluate them. We decided to split our data into a training set and a testing set that contains 80% and 20% of the data, respectively. We decided not to go with a validation set because we do not have enough data to split in a 80%, 10% and 10% manner for the training, testing and validation sets. As more data is made available and a better ground truth is constructed for further studies, a validation set should also be included. We tried out five models: random forests, SVM, KNN, XGBoost and neural networks. We selected these five models because they represent a mix of decision tree-based methods (Random Forest, XGBoost), distance-based classification (KNN), margin-based classification (SVM), and deep learning (Neural Networks). All the data has been normalized using a standard scalar. We used a grid

search on each model to find the best parameters for them using a 5-fold cross-validation. Below, we outline the models and the hyperparameters explored.

5.4.1 Random Forest

Random Forest is an ensemble learning method that constructs multiple decision trees and outputs the mode of the classes. The hyperparameters optimized include:

- n_estimators: Number of trees in the forest (values: 50, 100, 150).
- max_depth: Maximum depth of the trees (values: None, 10, 20, 30).
- min_samples_split: Minimum number of samples required to split an internal node (values: 2, 5, 10).
- min_samples_leaf: Minimum number of samples required to be at a leaf node (values: 1, 2, 4).

5.4.2 Support Vector Machines (SVM)

SVM is a powerful classification model that finds the hyperplane that best separates the data. The hyperparameters optimized include:

- C: Regularization parameter (values: 0.1, 1, 10).
- kernel: Specifies the kernel type (values: 'linear', 'rbf').
- gamma: Kernel coefficient (values: 'scale', 'auto').

5.4.3 K-Nearest Neighbors (KNN)

KNN is a non-parametric model that classifies a sample based on the majority class of its nearest neighbours. The hyperparameters optimized include:

- n_neighbors: Number of neighbors to consider (values: 3, 5, 7).
- weights: Weight function used in prediction (values: 'uniform', 'distance').
- metric: Distance metric used for neighbours (values: 'Euclidean', 'Manhattan').

5.4.4 XGBoost

XGBoost is a gradient-boosting algorithm known for its efficiency and performance. The hyperparameters optimized include:

- n_estimators: Number of boosting rounds (values: 50, 100, 150).
- max_depth: Maximum tree depth (values: 3, 5, 7).
- learning_rate: Step size shrinkage to prevent overfitting (values: 0.01, 0.1, 0.2).

5.4.5 Neural Networks

Neural networks capture complex patterns in data through layers of interconnected nodes. The hyperparameters optimized include:

- hidden_layer_sizes: Number of neurons per hidden layer (values: (50,), (100,), (50, 50)).
- activation: Activation function for hidden layers (values: 'relu', 'tanh').
- solver: Optimizer used for weight updates (values: 'adam', 'sgd').

5.5 Results and Discussion

5.5.1 Results

After performing a grid search for each model, we identified the optimal hyperparameters that yielded the best performance:

- **Random Forest**: Max depth = None, Min samples per leaf = 1, Min samples per split = 2, Number of estimators = 50.
- **SVM**: Regularization parameter $C = 10$, Gamma = `auto`, Kernel = `linear`.
- **KNN**: Metric = `euclidean`, Number of neighbors = 7, Weights = `distance`.

- **XGBoost**: Learning rate = 0.2, Max depth = 5, Number of estimators = 50.
- **Neural Network**: Activation function = `relu`, 2 hidden layer with 50 neurons in each, Solver = `adam`.

Using these optimal hyperparameters, we evaluated the models across multiple performance metrics, summarized in Table 5.2.

Metric	**Random Forest**	**SVM**	**KNN**	**XGBoost**	**Neural Networks**
Accuracy	0.96	0.88	0.92	0.92	0.92
Precision	0.96	0.88	0.93	0.93	0.93
Recall	0.96	0.88	0.92	0.92	0.92
F1-score	0.96	0.88	0.92	0.92	0.92
AUC-ROC	0.99	0.96	0.92	0.99	0.92
Log Loss	0.26	0.33	2.88	0.16	0.94
Inference Time (ms)	0.04	0.03	0.91	0.08	0.04
Model Size (MB)	0.18	0.08	0.15	0.20	0.30

Table 5.2: Evaluation Metrics for Machine Learning Models

5.5.2 Model Trade-Offs and Justifications

Table 5.2 summarizes the evaluation metrics for the five machine learning models. These metrics—accuracy, precision, recall, F1-score, AUC-ROC, log loss, inference time, and model size—highlight the strengths and weaknesses of each approach. The selection of the optimal model depends on the specific application requirements, balancing classification performance with probability calibration, inference speed, and storage constraints.

Overall Classification Performance

Random Forest achieves the highest overall accuracy at 96%, with precision, recall, and F1-score also reaching 96%. In contrast, SVM shows lower performance with an accuracy of 88%. KNN, XGBoost, and Neural Networks offer intermediate accuracy at 92%. Although Random Forest leads in raw classification accuracy, additional metrics must be considered for a holistic evaluation.

Discriminative Power (AUC-ROC) and Probability Calibration

The AUC-ROC curve graphically represents a classifier's performance by plotting the true positive rate against the false positive rate at various threshold settings, and its area under the curve (AUC) quantifies the model's overall ability to distinguish between the positive and negative classes [25].

In terms of AUC-ROC, both Random Forest and XGBoost perform exceptionally, each scoring 0.99, which indicates a strong ability to distinguish between classes. SVM follows with an AUC-ROC of 0.96, while KNN and Neural Networks lag slightly behind with scores of 0.92. When examining probability calibration via log loss, XGBoost outperforms the others with a low value of 0.16, followed by Random Forest at 0.26 and SVM at 0.33. In contrast, Neural Networks (0.94) and especially KNN (2.88) demonstrate poorer calibration. These results suggest that for applications where reliable probability estimates are critical—such as risk assessment—XGBoost would be particularly advantageous.

Inference Speed and Deployment Efficiency

For real-time applications, inference time is a crucial factor. SVM delivers the fastest inference at 0.03 ms, while Random Forest and Neural Networks are similarly efficient at 0.04 ms. XGBoost's inference time is slightly higher at 0.08 ms. However, KNN requires a significantly longer time of 0.91 ms, rendering it less suitable for latency-sensitive environments. Therefore, models such as SVM, Random Forest, and Neural Networks are more appropriate when rapid predictions are necessary.

Model Complexity and Storage Constraints

Storage efficiency is another important consideration, particularly for deployment on resource-limited devices. SVM is the most lightweight model at 0.08 MB. KNN and Random Forest require moderate storage at 0.15 MB and 0.18 MB, respectively, while XGBoost and Neural Networks are slightly larger at 0.20 MB and 0.30 MB. These differences may influence model selection in scenarios where memory is at a premium.

Final Model Selection

Taking all factors into account—classification accuracy, discriminative power, probability calibration, inference speed, and storage requirements—Random Forest and XGBoost

emerge as the strongest candidates. Random Forest offers excellent accuracy (96%), outstanding AUC-ROC (0.99), and rapid inference (0.04 ms) with a modest storage footprint (0.18 MB). XGBoost, while matching Random Forest in AUC-ROC and offering superior probability calibration (log loss of 0.16), has slightly lower accuracy (92%) and a marginally higher inference time (0.08 ms).

Given these trade-offs, we believe that Random Forest is the best model for our data. Its balanced performance across all key metrics makes it well-suited for scenarios requiring both high accuracy and real-time deployment. While XGBoost is attractive for its excellent calibration, the overall robustness and efficiency of Random Forest best meet our application's requirements.

5.5.3 Threats to Validity

One of the primary limitations of our machine learning model is the limited availability of labelled data. Our ground truth consists of only 126 scripts, as no prior studies have systematically classified behavioural biometrics scripts. Given this small dataset, there is a high likelihood that the models are overfitting to the training data.

To assess the extent of overfitting, we compared model performance on the training and test sets. We observed that all models achieved over 98% precision on the training set, whereas their performance on the test is on average 92% for all models. This discrepancy suggests that the models may have learned dataset-specific patterns rather than generalizable characteristics of behavioural biometrics scripts.

To mitigate overfitting, future research should focus on expanding the dataset by identifying and labelling additional scripts across a broader range of websites. Additionally, techniques such as data augmentation, synthetic data generation, and semi-supervised learning could be explored to improve model robustness.

This study primarily aimed to establish a machine learning framework for detecting behavioural biometrics scripts and provide a foundation for future research. To encourage further development, we have released our machine learning pipeline as an open-source tool, available at [8].

5.6 Conclusion

This chapter demonstrated that behavioural biometrics scripts are not the only type of scripts that produce taint flows, highlighting the necessity of a machine learning-based clas-

sification approach. We constructed a ground truth dataset by manually labelling scripts based on taint analysis reports and engineered features that capture key characteristics of script behaviour.

To evaluate different classification approaches, we tested five machine-learning models with hyperparameter tuning. Our analysis found that Random Forests provided the best balance between accuracy, precision, recall, and AUC-ROC, making it the most suitable model for our task.

While our results are promising, the study is limited by the size of the dataset, and additional data collection is necessary to improve model generalization. Future research can build upon this work by expanding the dataset, refining feature extraction techniques, and exploring deep-learning approaches for script classification.

Chapter 6

Results

6.1 Introduction

In the preceding chapters, we detailed the design and implementation of the various tools developed to detect behavioural biometrics scripts. This section describes how these tools are integrated into a unified framework for our analysis. We explain our methodology for selecting the websites and constructing the evaluation dataset, and we conclude with an overview of the key findings and insights derived from our experiments.

6.2 BBAT

Our analysis framework, called BBAT [7] (Behavioural Biometrics Analysis Tool), is an open-source tool that comprises three primary components:

1. **Dynamic Taint Analysis Browser**: This component is a customized version of Firefox, extended to track specific sources and sinks relevant to our study (a comprehensive list is provided in Appendix B).
2. **Custom Crawlers**: As described in the previous chapter, we developed a model that simulates user interactions on a webpage. All captured taint flow data is subsequently stored in MongoDB. We employ three distinct custom crawlers:
   (a) **Checkout Crawler**: This crawler is designed to navigate to checkout pages. Once the checkout page is reached, our user simulation module is executed.

(b) **Simple Crawler**: This crawler loads the given page of a website and runs the user simulation module. It is used for analyzing individual pages that are directly accessible via URL. (Note that checkout pages are not directly reachable by URL, as they typically require items to be present in a shopping cart.)

(c) **Login Crawler**: Developed by Tariq [49], this crawler locates the login page of a given website. The tool is based on SSOMonitor [29], extended to have multiple login candidates and does not rely on heuristics to find login pages. As it does not perform user simulation, the login page identified must be subsequently processed by the Simple Crawler for further analysis.

3. **Machine Learning Classifier**: The final component extracts the relevant taint flow data from MongoDB and transforms it into a data frame for analysis. Instead of relying on any single model that tends to overfit our data, as detailed in Chapter 5, we employ an ensemble approach using all five classifiers we trained. For each script, if any one of the models classifies it as a behavioural biometrics, the script is flagged as a potential behavioural biometrics. Although this strategy increases the number of false positives, it effectively minimizes false negatives. The resulting subset of scripts is then manually reviewed, significantly reducing the overall manual classification effort while ensuring that potential behavioural biometrics are not overlooked.

   To classify the collected scripts as either behavioural biometrics or non-behavioural biometrics, we employed the same approach as in Section 5.2.4. The scripts we were not able to classify will be left as potential behavioural biometrics scripts.

6.3 Analysis

Our analysis is structured around three studies that leverage the tools described earlier, particularly the dynamic taint analysis browser and the suite of custom crawlers. In each study, we apply our methodology to different datasets and page types to uncover insights about the deployment of behavioural biometrics scripts.

Study 1: Prevalence across the Web

In the first study, we assess the overall prevalence of behavioural biometrics scripts by examining a broad cross-section of the web. We applied our dynamic taint analysis browser and the simple crawler to the top 20,000 websites from Tranco's list [35]. The simple crawler

navigates to the main page of each site, where our user simulation module is executed to trigger potential script interactions. The taint flow data captured during these sessions is then stored in MongoDB for subsequent analysis by our machine learning classifier. This study establishes a baseline for how widespread behavioural biometrics scripting is in general web content.

Study 2: Analysis of Login Pages

The second study focuses on websites that typically handle sensitive data by analyzing login pages. We use a dataset comprising two sets: 500 banking websites and 500 random websites. The 500 banking websites dataset was created by Tariq [49] by finding all valid banks in the United States that have a website and selecting the first 500 banks. The 500 random websites list was created by using the first 500 entries that we successfully crawled from Study 1.

For each website in the two datasets, we initially used the simple crawler to obtain baseline data from the main page (this was done for both the 500 random websites and 500 bank websites). Next, we employ the login crawler to specifically target and retrieve the login pages from both sets. Once the login pages are found for both sets, we run our simple crawler on the login pages for both sets. The data from these targeted crawls is then compared to the baseline results to examine differences in the deployment of behavioural biometrics scripts between main pages and login pages.

Study 3: Focus on E-commerce and Checkout Pages

In the third study, our attention turns to the e-commerce sector. We applied our tool to two datasets: one consisting of the top 1,000 Shopify websites and another comprising 118 additional e-commerce sites, as described in the Checkout Crawler Chapter 4.

For these websites, we first use the simple crawler to get our baseline. Then, we use the checkout crawler to navigate to the checkout pages. Upon reaching a checkout page, the user simulation module is activated, and all subsequent taint flow data is captured and stored. This study allows us to investigate the prevalence of behavioural biometrics scripts in the context of e-commerce and to examine any differences between main e-commerce pages and checkout-specific interactions.

Across all studies, the data gathered by the dynamic taint analysis browser and the respective crawlers is processed by our machine learning classifier. This final step predicts

whether each script implements behavioural biometrics, enabling us to conclude both the prevalence and the context of deployment for these scripts.

6.4 Results

This section will show the results of the crawl and the efficiency of the crawler. The behavioural biometrics results will be found for all studies at the end of this section.

6.4.1 Study 1

Study 1 is on the tranco top 20k list. Table 6.1 shows the status messages that were encountered during this study. Completed refers to no error and that the crawl was successful, while all the other messages are a type of error message that was encountered. This table only has the error messages for reoccurring errors that appear more than 10 times. Other errors that are not common do not appear in this table.

Crawl Message	**Number**
Completed	13,278
page.goto: NS_ERROR_UNKNOWN_HOST	3,207
page.goto: Timeout 30000ms exceeded.	1,660
HTTP error 403	597
HTTP error 404	332
page.goto: Navigation failed because page crashed!	199
page.goto: NS_ERROR_CONNECTION_REFUSED	192
page.evaluate: Target closed	51
page.goto: SSL_ERROR_UNKNOWN	74
page.goto: Navigation failed because page was closed!	40
HTTP error 503	37
page.goto: NS_ERROR_NET_RESET	23
page.evaluate: can't access property "click", document.body is null	23
page.goto: NS_ERROR_NET_INTERRUPT	20
HTTP error 400	12
HTTP error 401	11
HTTP error 500	11

Table 6.1: Common crawl messages and their counts for the top 20k websites

6.4.2 Study 2

Main Pages

Study 2 is on the login pages. Since we took only the top 500 successful crawls to form the dataset of random websites, there is no status message to look at for that dataset. The most common crawl messages for the 500 bank websites' main page can be found in Table 6.2.

Crawl Message	Number
Completed	462
page.goto: Timeout 30000ms exceeded.	12
HTTP error 403	27
page.evaluate: can't access property "indexOf", type is undefined	5
page.goto: NS_ERROR_NET_RESET	2
page.goto: NS_ERROR_CONNECTION_REFUSED	2
page.goto: SSL_ERROR_UNKNOWN	1

Table 6.2: Common crawl messages and their counts for the 500 bank websites main page

Login Pages

We took the two datasets and we ran the login crawler. This login crawler will find all login candidates. So each website can have more than one login page. The login crawler gave us 1355 login pages for the bank websites and 1489 login pages for the 500 random websites. This outcome occurs because individual websites can host multiple login pages, and the observed difference in the number of login pages is due to the different design choices made by each website. The most common crawl messages for the 1355 bank websites' login pages can be found in Table 6.3 and the 1489 login pages for the random websites can be found in Table 6.4.

6.4.3 Study 3

Main Pages

Study 3 is on e-commerce pages. Tables 6.5 and 6.6 show the most prevalent status messages of the e-commerce main pages.

Crawl Message	Number
Completed	1315
page.goto: Timeout 30000ms exceeded.	16
page.goto: Navigation failed because page crashed!	4
page.evaluate: can't access property "indexOf", type is undefined	7
HTTP error 404	4
HTTP error 500	2
HTTP error 401	2

Table 6.3: Common crawl messages and their counts for the 1355 bank websites login page

Crawl Message	Number
Completed	1265
page.goto: Timeout 30000ms exceeded.	91
HTTP error 400	75
keyboard.press: Target page, context or browser has been closed	34
page.goto: Navigation failed because page crashed!	8
page.goto: NS_ERROR_ABORT	5
HTTP error 404	3

Table 6.4: Common crawl messages and their counts for the 1489 random websites login page

Checkout Pages

To study the behaviour on the checkout pages, we used the checkout crawler mentioned previously. We only crawled the websites that we were able to successfully crawl the main page in the first part of the study. We used all four of the checkout modules we have and only captured taint information once we hit the checkout page. Once a checkout page has been reached, the rest of the pages associated with that website will no longer be crawled and the status of that website will be marked as done. Since the checkout crawler has four modules and each module checks more than one webpage, we cannot use status messages for the efficiency of the crawler. Instead, we use the number of checkout pages reached as a successful crawl and the rest of the pages as an unsuccessful crawl, which can be found in Table 6.7.

Crawl Message	Number
Completed	897
page.goto: Timeout 30000ms exceeded.	57
page.goto: NS_ERROR_UNKNOWN_HOST	23
page.goto: Navigation failed because page crashed!	8
page.goto: Navigation failed because page was closed!	4
page.evaluate: Execution context was destroyed	3
page.evaluate: Target closed	2

Table 6.5: Common crawl messages and their counts for the top 1000 Shopify websites

Crawl Message	Number
Completed	104
page.goto: Timeout 30000ms exceeded.	6
page.goto: NS_ERROR_UNKNOWN_HOST	2
HTTP error 404	1
HTTP error 403	1
page.goto: NS_ERROR_NET_RESET	1
page.goto: Navigation failed because page crashed!	1

Table 6.6: Common crawl messages and their counts for 118 e-commerce websites

6.4.4 Behavioural Biometrics Scripts

We processed all the data from MongoDB for each dataset in each study. We then ran both our machine learning classifier and our manual classification, as discussed previously. Table 6.8 presents the behavioural biometrics data for each study and run.

Since the number of websites analyzed differs for each crawl, we created Table 6.9. This table displays the percentage of behavioural biometrics scripts and the percentage of websites that employ behavioural biometrics, calculated based on the number of scripts and websites visited per crawl. This approach allows us to compare different crawls, even when they involve different sets of websites.

6.5 Discussion

After running our tool, we are now able to address the research questions posed in this thesis. Additionally, we made several other noteworthy observations throughout this research,

Crawl Status	E-commerce	Shopify
Pages Crawled	104	897
Successful Crawl	41	507
Unsuccessful Crawl	63	390

Table 6.7: E-commerce and Shopify checkout page status

which will also be discussed in this section.

6.5.1 Research Questions

Prevalence: How widespread are behavioural biometrics scripts on the web? (RQ1)

We examine the results of Study 1 (see Table 6.8 and 6.9). Our analysis reveals that less than 1% of the successfully crawled websites contained any form of behavioural biometrics script, indicating that these scripts are not yet widespread. However, their presence confirms that they are in active use. Although currently uncommon, we believe that behavioural biometrics scripts are in an early stage of adoption and expect their prevalence to grow over time.

Sensitive Pages: Are these scripts disproportionately deployed on pages handling sensitive data (e.g., logins and checkout pages)? (RQ2)

Table 6.9 shows that sensitive pages tend to contain a higher number of behavioural biometrics scripts. For example, many bank websites deploy scripts that activate only on the login page. Similarly, for Shopify websites, a Shopify-specific script appears to load on the checkout page, which we suspect to be a behavioural biometrics script. This script was flagged by our machine learning classifier and appears on around half of the Shopify checkout pages. More information about this script can be found in subsection 6.5.1 Therefore, our findings indicate that behavioural biometrics scripts are more likely to be deployed on sensitive pages such as login or checkout pages.

Crawl	Scripts with Taint Flows	Scripts detected by Classifier	Confirmed BB Scripts	Potential BB Scripts	Websites Analyzed	Websites Confirmed BB	Websites Potential BB
Study 1: Top 20k Crawl	2913	652	50	36	13,278	41	25*
Study 2: 500 Bank Main Page Crawl	63	15	4	0	462	3	0
Study 2: 500 Random Main Page Crawl	163	66	6	13	500	3	8*
Study 2: 500 Bank Login Page Crawl	191	48	31	0	462	21	0
Study 2: 500 Random Login Page Crawl	206	84	16	15	500	5	10
Study 3: 1000 Shopify Main Page Crawl	661	392	1	0	897	1	0
Study 3: 118 E-Commerce Main Page Crawl	52	23	1	0	104	1	0
Study 3: 1000 Shopify Checkout Page Crawl	758	407	0	324	507	0	188
Study 3: 118 E-Commerce Checkout Page Crawl	31	12	0	9	41	0	4

* Many of these websites are different regional Amazon websites that use the same script.

Table 6.8: Behavioural Biometrics scripts for all studies

Login vs. Main Pages: Are login pages more likely to host behavioural biometrics scripts than main pages? (RQ3)

When comparing main pages to login pages, a similar trend emerges: login pages host a greater number of behavioural biometrics scripts. This pattern is observed in both banking websites and random websites, where login pages exhibit an increased count of detected behavioural biometrics scripts and confirmed websites using the technology.

Banking Logins: Do banking login pages employ these scripts more frequently than non-banking logins? (RQ4)

Although random pages tend to have more behavioural biometrics scripts on their login pages, the increase is not as pronounced as that observed for bank login pages. As shown in

Crawl	% BB Script out of Tracking Scripts	% Potential BB out of Tracking Scripts	% Websites with BB	% Websites with Potential BB
Study 1: Top 20k Crawl	1.71%	1.24%	0.31%	0.19%
Study 2: 500 Bank Main Page Crawl	6.35%	0.00%	0.65%	0.00%
Study 2: 500 Random Main Page Crawl	3.68%	7.98%	0.65%	0.16%
Study 2: 500 Bank Login Page Crawl	16.23%	0.00%	4.55%	0.00%
Study 2: 500 Random Login Page Crawl	7.76%	7.28%	1.00%	2.00%
Study 3: 1000 Shopify Main Page Crawl	0.15%	0.00%	0.11%	0.00%
Study 3: 118 E-Commerce Main Page Crawl	1.92%	0.00%	0.96%	0.00%
Study 3: 1000 Shopify Check-out Page Crawl	0.00%	42.74%	0.00%	37.08%
Study 3: 118 E-Commerce Checkout Page Crawl	0.00%	29.03%	0.00%	9.76%

Table 6.9: Behavioural Biometrics scripts for all studies

Table 6.9, the number of confirmed bank websites employing behavioural biometrics rises from 3 on main pages to 21 on login pages, while the corresponding jump for random pages is only from 3 to 5. This suggests that entities handling highly sensitive information, such as banks, are more likely to deploy advanced protection techniques on their login pages.

Checkout Pages: Are checkout pages more likely to deploy behavioural biometrics scripts? (RQ5)

Answering this question is challenging because most of the scripts identified on checkout pages could not be definitively confirmed as behavioural biometrics scripts. Although we are inclined to believe they perform behavioural biometrics, there is less certainty compared to scripts that explicitly claim to implement such techniques. Based on the

available evidence, it is highly likely that these checkout page scripts are intended to serve a behavioural biometrics function.

Another interesting observation is that many of the tracking scripts (confirmed not to be behavioural biometrics scripts) that were present on the main page were absent on the checkout page. This finding reinforces the idea that sensitive pages, such as checkout pages, are less likely to include tracking scripts for marketing or analytics purposes, likely as a measure to enhance user privacy and security.

Shopify Support: Does Shopify, the leading e-commerce platform, enable behavioural biometrics scripts? (RQ6)

Although we cannot conclusively verify that the checkout script found on the Shopify website qualifies as a behavioural biometrics script, the evidence strongly suggests that Shopify employs such scripts to prevent fraud. This is because the checkout script is loaded exclusively on the checkout page, is not registered with any recognized analytics provider, and utilizes sources and sinks that are consistent with those observed in other behavioural biometrics scripts. We also could not find any documentation on Shopify's website discussing these scripts.

Interestingly, these scripts appear exclusively on checkout pages. Notably, it is present on almost half of the checkout pages we analyzed. However, it is possible that some checkout pages do not activate the script, which could explain why not every Shopify page displayed it during our manual investigation. Interestingly, we identified one confirmed behavioural biometrics script on a non-checkout page, demonstrating that behavioural biometrics are not strictly confined to Shopify checkout pages.

6.5.2 Extra Findings

Tracking Script Prevalence

In Table 6.9, we observe that only a very small percentage of scripts tracking user behaviour (such as mouse movements and keystrokes) are classified as behavioural biometrics scripts. This indicates that most scripts exfiltrating this information are not primarily intended for security purposes. In our analysis, we found that many of these scripts are associated with heatmaps or session replays, while a significant portion are web analytics scripts that also capture user behaviour data. Although the exact purpose behind the collection of

this sensitive information in web analytics scripts remains unclear, we believe that greater transparency regarding its use should be provided to users.

Another interesting finding is the variation in the prevalence of behavioural biometrics scripts across different types of websites. In Table 6.9, we can see that among the 500 bank websites examined, 6.35% incorporated behavioural biometrics scripts, whereas only 0.15% of the 1,000 main Shopify e-commerce pages did. For the top 20k websites, the average prevalence was 1.65%. These results suggest that behavioural biometrics scripts occur more frequently on bank websites, which we believe is due to the sensitive nature of their operations. Conversely, e-commerce pages exhibit a higher volume of tracking scripts with an analytical purpose, indicating that the tracking mechanisms on banking websites are generally more secure and less likely to be malicious.

Crawler Effectiveness

In Table 6.1, we observe that the crawler succeeded on 13,278 out of the 20,000 websites visited. Notably, many errors were not caused by the crawler itself; for example, 3,207 websites returned an "unknown host" error and 597 resulted in an HTTP 403 error. These issues suggest that the Tranco list [35] may be outdated, highlighting the need for a refreshed top websites list.

Overall, we estimate that approximately 2,000 websites failed due to limitations of our tool, yielding an estimated success rate of around 90%. This rate is corroborated by the results from the Shopify main pages (Table 6.5) and bank main pages (Table 6.2), where the crawler achieved a similar success rate. The majority of the errors were caused by FoxHound crashes, an area that presents an opportunity for future improvement by looking at why the crashes occur and finding ways to fix them. Considering the pioneering nature of this study, a 90% success rate is a reasonable outcome.

Additionally, the successful crawls for the checkout pages on both e-commerce and Shopify websites (39% and 57% respectively), as detailed in Table 6.7, are considerably lower than those observed in our overall analysis in Chapter 4 (54% and 78% respectively). This discrepancy is likely due to the compounding effects of FoxHound errors combined with the challenges of navigating multiple pages and steps within each website. Each additional navigation step introduces another potential point of failure, hindering progress toward reaching the checkout page. Addressing the FoxHound crashes should, therefore, lead to a significant improvement in the overall number of successful crawls.

Potential Behavioural Biometrics

In Study 1, we identified 25 websites with scripts that could not be clearly classified as either behavioural biometrics or web analytics. We labelled these as potential behavioural biometrics scripts. Notably, 14 of these websites were Amazon domains from various regions. All Amazon websites deployed identical scripts that collected the same information and operated in the same manner. A similar pattern emerged in Study 2, which involved a crawl of 500 random main pages: out of the eight potential behavioural biometrics scripts detected, seven belonged to Amazon websites. This concentration on Amazon sites is an important consideration, as it may skew the overall results. We have taken this factor into account in our previous analysis and when addressing the research questions.

The remaining potential behavioural biometrics scripts in Study 1 were associated with Alibaba and two other unidentified companies. The Alibaba scripts, in particular, did not provide sufficient evidence to classify them as analytic scripts. They also featured the keywords `SecDev` and `SecuritySDK` in their names, suggesting that these scripts might be used for non-malicious purposes. Overall, these Alibaba scripts were deployed across nine different websites in Study 1 and one website in Study 2.

New Behavioural Biometrics Scripts

While manually classifying all scripts across our studies, we discovered four companies that offer innovative behavioural biometrics solutions not previously recognized in our research. Below is a summary of these companies and their offerings, *as claimed on their websites:*

- **Yofi**: Yofi leverages real-time alerting and behavioural biometrics signals to proactively identify high-risk individuals and uncover hidden user clusters. Their solution is designed to integrate seamlessly into existing systems without disrupting the experience of legitimate users.
- **Cheq**: Cheq has developed a detection engine that combines multiple factors (including behavioural biometrics) to detect and mitigate both automated and human-driven threats. This multi-layered strategy is focused on preserving customer trust by preventing fraudulent activities.
- **Group-IB**: A cybersecurity provider, Group-IB incorporates behavioural biometrics techniques as part of its comprehensive anti-fraud and brand protection solutions. By integrating advanced threat intelligence with behavioural analysis, they address increasingly sophisticated cyberattacks.

- **Untarget.ai**: A Russian company specializing in real-time analysis of website visitor behaviour, Untarget.ai focuses on detecting click fraud. Their system continuously monitors user interactions and dynamically adjusts client account settings to mitigate the impact of fraudulent clicks on advertising budgets.

6.6 Conclusion

In this chapter, we took all the parts that we created in previous chapters and built BBAT [7], which is our open-source behavioural biometrics analysis tool. We then created three sets of studies to help us answer our research questions. Our tool was used for all three studies alongside some manual classification to get the number of behavioural biometrics scripts on the web. We were able to conclude that there are more behavioural biometrics scripts on sensitive pages like login and checkout. We also saw that banking websites have fewer tracking scripts and a higher percentage of behavioural biometrics scripts, which is the opposite of what we noticed for e-commerce websites. We also found four new behavioural biometrics companies that were previously unknown to us. We also pinpointed the two biggest areas of improvement, which are generating a more comprehensive ground truth and fixing the bugs that are making FoxHound crash on some pages.

Chapter 7

Conclusion

7.1 Contributions

This thesis bridges a critical gap in the detection and analysis of behavioural biometrics scripts by integrating three interdependent innovations into a comprehensive framework. Our contributions include:

- **Enhanced Dynamic Taint Analysis**: By extending FoxHound with support for boolean tainting, inter-process communication tracking, and resilience to obfuscation, we achieved a 97% effectiveness rate in identifying behavioural biometrics data flows. This advancement enables the first open-source framework capable of tracing fine-grained user interactions (e.g., keystrokes, mouse movements) to their corresponding exfiltration points.
- **Checkout Crawler**: We developed a novel crawler that simulates realistic user interactions to navigate e-commerce websites. With an accuracy of 78%, this first-of-its-kind crawler unlocks the possibility to analyze checkout pages, which are sensitive pages that have previously never been looked at in studies.
- **Machine Learning Classifier:** Our Random Forests classifier, trained on 126 labelled scripts, achieved 95.7% accuracy in differentiating behavioural biometrics scripts from session replay and heatmap scripts.

Collectively, these components form the *Behavioral Biometrics Analysis Tool (BBAT)*, a scalable, automated solution for the large-scale detection and analysis of the behavioural biometrics script.

7.2 Key Findings

Deploying BBAT across 20,000 websites, 1,000 login pages, and over 1,000 e-commerce websites produced several noteworthy insights:

- **Prevalence:** Behavioral biometrics scripts are used by 0.3% of top websites. The percentage jumps to 4.55% when looking at bank login pages. This technology is still new and not utilized by many websites, but we do believe there will be an increase in adoption.
- **Sensitive Pages**: Sensitive pages like login or checkout pages are more likely to contain behavioural scripts. These pages tend to also have fewer tracking scripts on them that are analytics or session replay. This explains why 4.55% of bank login pages had this technology on them.
- **Shopify Websites**: Shopify websites employ a high number of tracking scripts on their pages. However, most of those tracking scripts are not employed once the checkout page is reached. A new checkout-specific script is loaded, which we believe is behavioural biometrics.
- **New Scripts**: While doing the manual classification, we were able to find four new behavioural biometrics companies and the scripts associated with them.

7.3 Recommendations for Ethical Deployment

To mitigate privacy risks while harnessing the security benefits of behavioural biometrics, we propose a multi-faceted approach that encompasses legal, technical, and procedural measures. These recommendations are designed to ensure that the deployment of behavioural biometrics systems is both ethically sound and compliant with emerging regulatory frameworks.

1. **Transparency Mandates**:
    - Disclosure Requirements: Companies should be legally obligated to explicitly disclose the use of behavioural biometrics in their privacy policies. Such disclosures must detail the types of data collected (e.g., keystroke dynamics, mouse movement patterns, touch pressure), the specific purposes for which the data

is used (e.g., fraud detection, continuous authentication), and any third parties with whom the data is shared. This transparency not only builds user trust but also facilitates independent scrutiny of biometrics practices.

- Obfuscation Bans: Regulatory bodies (e.g., under GDPR, CCPA) should impose restrictions on the excessive obfuscation of behavioural scripts, such as heavy minification or encryption that conceals the system's inner workings. This would allow third-party auditors and researchers to verify that biometrics systems adhere to declared practices and that no hidden data-sharing or tracking mechanisms exist. From what we noticed, the main logic happens in the backend and these scripts mostly collect the data, which in our opinion is not enough justification to obfuscate these scripts.

2. **Privacy-by-Design Practices**:

   - Client-Side Anonymization: To minimize privacy risks, raw interaction data should be processed locally on the client's device. For example, instead of transmitting exact keystroke timings or precise mouse coordinates, systems should compute anonymized features (such as entropy scores or aggregated metrics) that are less sensitive to individual identification while still being useful for authentication. We noticed that some scripts employ these methods but not all of them.
   - Opt-Out Mechanisms: Users must be provided with clear, accessible controls to opt out of behavioural tracking. These mechanisms should be as prominent and intuitive as cookie consent banners, ensuring that users can disable biometrics tracking without experiencing a degradation in service quality.
   - Data Minimization and Retention: Systems should adhere to the principles of data minimization by collecting only the necessary behavioural data for authentication purposes. Additionally, any biometrics data collected should have clearly defined retention periods after which the data is securely deleted or anonymized.

3. **Standardized Evaluation:**

   - Third-Party Audits: Regular, independent audits by certified third-party organizations should be mandated. These audits would assess not only the security robustness of biometrics systems but also their compliance with privacy standards and ethical guidelines. The audit reports should be made publicly available to foster accountability and continuous improvement.

7.4 Future Directions

This work lays a robust foundation for several promising research avenues, which can further advance the field of behavioral biometrics detection and analysis. We outline several potential directions for future research:

- Longitudinal Studies: Future research should monitor adoption trends post-regulation, assessing both compliance and the evolution of evasion techniques over time. Longitudinal studies can provide valuable insights into how regulatory measures impact the deployment and modification of behavioural biometrics systems and reveal adaptive strategies used by adversaries.
- Cross-Device and Cross-Platform Analysis: Extending our detection framework to mobile applications, IoT devices, and emerging platforms is essential. These platforms increasingly employ diverse biometrics modalities (e.g., touch dynamics, voice, and gesture recognition) that may require novel taint analysis techniques and machine learning approaches to capture and interpret behavioural patterns accurately.
- Ground Truth Formation: A major limitation in current research is the scarcity of comprehensive, high-quality datasets. Future work should focus on collecting and curating a larger and more diverse corpus of behavioural biometrics scripts and websites that deploy them. Establishing a robust ground truth dataset will facilitate improved training, benchmarking, and validation of machine learning models, ultimately enhancing detection accuracy and fairness.
- Privacy-Enhancing Technologies: In parallel with advancing detection capabilities, it is critical to explore methods that preserve user privacy. Future studies could investigate the application of privacy-preserving techniques such as homomorphic encryption, differential privacy, or federated learning. These approaches would allow the secure processing of biometrics data without compromising the system's ability to detect fraud or anomalies.

Behavioural biometrics represent a double-edged sword: their capacity to prevent fraud is substantial, yet their covert operation may break digital trust if misused. This thesis advocates for a balance between enhanced security and user privacy by emphasizing transparency, accountability, and user empowerment. As authentication evolves from traditional passwords to behaviour-based mechanisms, it is imperative that regulators, researchers, and industry stakeholders vigilantly guard the boundary between improved security and company surveillance. This will ensure that behavioural biometrics technologies are integrated ethically and improve user's trust in the web.

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*, CCS '14, page 674–689, New York, NY, USA, 2014. Association for Computing Machinery.

[2] Aftership. top 1000 shopify website. https://www.aftership.com/store-list/top-1000-shopify-stores.

[3] Ahmed Awad E. Ahmed and Issa Traore. A new biometric technology based on mouse dynamics. *IEEE Transactions on Dependable and Secure Computing*, 4(3):165–179, 2007.

[4] Amazon. What is ecommerce? https://sell.amazon.com/learn/what-is-ecommerce.

[5] Nampoina Andriamilanto and Tristan Allard. Brfast: a tool to select browser fingerprinting attributes for web authentication according to a usability-security trade-off. In *Companion Proceedings of the Web Conference 2021*, WWW '21, page 701–704, New York, NY, USA, 2021. Association for Computing Machinery.

[6] Nampoina Andriamilanto, Tristan Allard, Gaëtan Le Guelvouit, and Alexandre Garel. A large-scale empirical analysis of browser fingerprints properties for web authentication. *ACM Trans. Web*, 16(1), September 2021.

[7] Alexandru Bara. BBAT: Behavioural biometrics analysis tool. https://github.com/alexbara2000/BBAT.

[8] Alexandru Bara. Behavioural biometrics machine learning pipeline. https://github.com/alexbara2000/BBML.

[9] Alexandru Bara. Behavioural biometrics optimized foxhound version. https://github.com/alexbara2000/project-foxhound/tree/testBranch.

[10] Alexandru Bara. Checkout crawler. https://github.com/alexbara2000/checkout-crawler.

[11] berstend. puppeteer extra plugin stealth. https://github.com/berstend/puppeteer-extra/tree/master/packages/puppeteer-extra-plugin-stealth.

[12] Soumaya Boussaha, Lukas Hock, Miguel Bermejo, Rubén Cuevas Rumín, Ángel Cuevas Rumín, 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. *Proc. Priv. Enhancing Technol.*, 2024(3):540–560, 2024.

[13] BuiltWith. top 200 canadian e-commerce websites. https://builtwith.com/top-sites/Canada/eCommerce.

[14] Quan Chen and Alexandros Kapravelos. Mystique: Uncovering information leakage from browser extensions. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 1687–1700. Association for Computing Machinery.

[15] Software Freedom Conservancy. Selenium. https://www.selenium.dev. Accessed: 2025-03-20.

[16] Valter Crescenzi, Paolo Merialdo, and Disheng Qiu. Automatic data extraction from large websites. *VLDB*, 2011.

[17] DeepSource. Taint analysis. https://deepsource.com/glossary/taint-analysis.

[18] DuckDuckGo. Tracker radar collector. https://github.com/duckduckgo/tracker-radar-collector. Accessed: 2025-03-21.

[19] 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*, page 237–257, Berlin, Heidelberg, 2021. Springer-Verlag.

[20] Peter Eckersley. How unique is your web browser? In Mikhail J. Atallah and Nicholas J. Hopper, editors, *Privacy Enhancing Technologies*, pages 1–18. Springer.

[21] 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*, CCS '16, pages 1388–1401. Association for Computing Machinery.

[22] R. Stockton Gaines, William Lisowski, S. James Press, and Norman Shapiro. *Authentication by Keystroke Timing: Some Preliminary Results*. RAND Corporation, Santa Monica, CA, 1980.

[23] GDPR. Complete guide to gdpr compliance. https://gdpr.eu. Accessed: 2025-03-20.

[24] geeksforgeeks. Explain the architecture of mozilla firefox browser. https://www.geeksforgeeks.org/explain-the-architecture-of-mozilla-firefox-browser/.

[25] Google. Classification: Roc and auc. https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc.

[26] Sandeep Gupta, Attaullah Buriro, and Bruno Crispo. A chimerical dataset combining physiological and behavioral biometric traits for reliable user authentication on smart devices and ecosystems. *Data in Brief*, 28:104924, 2020.

[27] Tommaso Innocenti, Matteo Golinelli, Kaan Onarlioglu, Ali Mirheidari, Bruno Crispo, and Engin Kirda. Oauth 2.0 redirect uri validation falls short, literally. In *Proceedings of the 39th Annual Computer Security Applications Conference*, ACSAC '23, page 256–267, New York, NY, USA, 2023. Association for Computing Machinery.

[28] 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, 2021.

[29] Louis Jannett, Maximilian Westers, Tobias Wich, Christian Mainka, Andreas Mayer, 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 (EuroS&P)*, pages 173–192. IEEE.

[30] Rahul Kanyal and Smruti R. Sarangi. PanoptiChrome: A modern in-browser taint analysis framework. In *Proceedings of the ACM on Web Conference 2024*, pages 1914–1922. ACM.

[31] Rezwana Karim, Frank Tip, Alena Sochůrková, and Koushik Sen. Platform-independent dynamic taint analysis for JavaScript. 46(12):1364–1379. Conference Name: IEEE Transactions on Software Engineering.

[32] David Klein, Thomas Barber, Souphiane Bensalim, Ben Stock, and Martin Johns. Hand sanitizers in the wild: A large-scale study of custom javascript sanitizer functions. In *Proc. of the IEEE European Symposium on Security and Privacy*, June 2022.

[33] David Klein, Marius Musch, Thomas Barber, Moritz Kopmann, and Martin Johns. Accept all exploits: Exploring the security impact of cookie banners. In *Proceedings of the 38th Annual Computer Security Applications Conference*, 2022.

[34] Pierre Laperdrix, Nataliia Bielova, Benoit Baudry, and Gildas Avoine. Browser fingerprinting: A survey. *ACM Trans. Web*, 14(2), April 2020.

[35] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. Tranco: A research-oriented top sites ranking hardened against manipulation. In *Proceedings of the 26th Annual Network and Distributed System Security Symposium*, NDSS 2019, February 2019.

[36] Sebastian Lekies, Ben Stock, and Martin Johns. 25 million flows later: large-scale detection of DOM-based XSS. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, CCS '13, pages 1193–1204. Association for Computing Machinery.

[37] Xu Lin, Panagiotis Ilia, Saumya Solanki, and Jason Polakis. Phish in sheep's clothing: Exploring the authentication pitfalls of browser fingerprinting. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1651–1668, Boston, MA, August 2022. USENIX Association.

[38] Jonathan R Mayer. "any person... a pamphleteer:" internet anonymity in the age of web 2.0.

[39] MDM web docs. Mdm web docs. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference.

[40] Microsoft. Playwright. https://playwright.dev. Accessed: 2025-03-20.

[41] Keaton Mowery and Hovav Shacham. Pixel perfect: Fingerprinting canvas in HTML5. In Matt Fredrikson, editor, *Proceedings of W2SP 2012*. IEEE Computer Society, May 2012.

[42] Nick Nikiforakis, Alexandros Kapravelos, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *2013 IEEE Symposium on Security and Privacy*, pages 541–555, 2013.

[43] Łukasz Olejnik, Gunes Acar, Claude Castelluccia, and Claudia Diaz. The leaking battery. In Joaquin Garcia-Alfaro, Guillermo Navarro-Arribas, Alessandro Aldini, Fabio Martinelli, and Neeraj Suri, editors, *Data Privacy Management, and Security Assurance*, pages 254–263, Cham, 2016. Springer International Publishing.

[44] Toshiharu Samura and Haruhiko Nishimura. Keystroke timing analysis for individual identification in japanese free text typing. In *2009 ICCAS-SICE*, pages 3166–3170, 2009.

[45] 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*, WWW '24, page 1690–1701, New York, NY, USA, 2024. Association for Computing Machinery.

[46] Chao Shen, Zhongmin Cai, Xiaomei Liu, Xiaohong Guan, and Roy A. Maxion. Mouseidentity: Modeling mouse-interaction behavior for a user verification system. *IEEE Transactions on Human-Machine Systems*, 46(5):734–748, 2016.

[47] Vladislav Shkapenyuk and Torsten Suel. Design and implementation of a high-performance distributed web crawler. *IEEE Data Eng. Bull.*, 2002.

[48] Alexander Sjösten, Steven Van Acker, and Andrei Sabelfeld. Discovering browser extensions via web accessible resources. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, CODASPY '17, page 329–336, New York, NY, USA, 2017. Association for Computing Machinery.

[49] Aswad Tariq. Login crawler. https://github.com/aswad546/SSO-Monitor.

[50] Pin Shen Teh, Andrew Beng Jin Teoh, and Shigang Yue. A survey of keystroke dynamics biometrics. *ScientificWorldJournal*, 2013:408280, November 2013.

[51] Princy Ann Thomas and K Preetha Mathew. A broad review on non-intrusive active user authentication in biometrics. *Journal of Ambient Intelligence and Humanized Computing*, 14(1):339–360, January 2023.

[52] Minh Tran, Xinshu Dong, Zhenkai Liang, and Xuxian Jiang. Tracking the trackers: Fast and scalable dynamic analysis of web content for privacy violations. In Feng Bao, Pierangela Samarati, and Jianying Zhou, editors, *Applied Cryptography and Network Security*, pages 418–435. Springer.

[53] Thomas Unger, Martin Mulazzani, Dominik Frühwirt, Markus Huber, Sebastian Schrittwieser, and Edgar Weippl. Shpf: Enhancing http(s) session security with browser fingerprinting. In *2013 International Conference on Availability, Reliability and Security*, pages 255–261, 2013.

[54] Boya Zhang. Analytic analysis scripts. https://github.com/boyazhang314/Fall-2024-URA.

APPENDICES

Appendix A

Scripts

Websites			
kaleido.ai	hostinger.com	actualicese.com	creativemarket.com
blogspot.com	nckls.org	maisfontes.com	colorlib.com
aspireapp.com	laposte.fr	exame.com	pluralsight.com
5thandglow.com	flyfrontier.com	trotec.com	detik.com
asics.com	dupe.com	loomly.com	kolosek.com

Table A.1: Websites containing various analytic scripts.

A.1 Web Analytics Scripts

In order to build our ground truth for the machine learning classifier, we used a list of 20 websites that contain analytic scripts. The websites have been found by Zhang [54]. The list can be found in Table A.1.

A.2 Behavioural Biometric Scripts

In joint work with Aswad Tariq, we were able to find scripts for 19 behavioural biometrics companies. Table A.2 shows the name of the company, alongside some of the websites that have these scripts. It is important to note that just because the website has the script does not necessarily mean the script is actively being used.

Provider	Websites
BioCatch	bmo.com tsb.co.uk laybuy.com citi.com hangseng.com itau.co openbank.es bankofamerica.com hsbc.com.hk td.com lloydsbank.co.uk halifax-online.co.uk natwest.com bankofscotland.co.uk icicibank.com natwest.com anz.com commbank.com.au desjardins.com synchronybank.com suncorpbank.com.au bendigobank.com.au nab.com.au mtb.com nationwide.co.uk mercurycards.com bmo.com barclays.co.uk mbna.co.uk westpac.com.au macquarie.com.au

Provider	Websites
Iovation (TransUnion)	stake.com upwork.com usbank.com jcpenney.com expedia.ca neofinancial.com starbucks.ca zoosk.com frankie4.com appliancesonline.com.au beautyheaven.com.au vortexoptics.com blackdiamondequipment.com leupold.com viewsonic.com eharmony.ca classmates.com elitepartner.de parship.de rab.equipment annsummers.com josephjoseph.com meccabingo.com flirt4free.com jacamo.co.uk jdwilliams.co.uk simplybe.co.uk creditonebank.com midcountry.bank secureinternetbank.com westernunion.com

Provider	Websites
Nudata	istockphoto.com singaporeair.com chime.com quill.com earny.co ipko.pl hrblock.com
Neuro-Id	biltrewards.com joracredit.com coign.com
Behaviosec (LexisNexis ThreatMetrix Module)	cibc.com royalbank.com marcus.com johnvarvatos.com tsb.co.uk cinemark.com.bo simplii.com citbank.com healthsafe-id.com avon.com booking.com bnpparibas.com sc.com
Sardine.ai	evergreenmoney.com wealthsimple.com blockchain.com stockx.com wealthsimple.com dankbank.co
Forter	upwork.com citizenwatch.com wish.com fiverr.com
PingIdentity	victoriassecret.com clareityiam.net

Provider	Websites
Accertify	jcpenney.com mytrip.com grubhub.com starbucks.com
Bureau	bureau.id
f5	cra.com* target.com
TypingDNA	typingdna.com
Human (PerimeterX)	fanduel.ca
ThreatMark	sparkasse.at
Transmit Security	ethoslife.com illimitybank.com
Darwinium	asaas.com
Feedzai	cajamar.es kutxabank.es
DataVisor	airasia.com
CallSign	wise.com sainsburysbank.co.uk
DataDome	shutterstock.com

Table A.2: Behavioural biometrics companies and websites that contain their scripts

* Login page cannot be reached via URL.

Appendix B

Dynamic Taint Analysis

B.1 Behavioural Biometrics Sources and Sinks

Table B.1 shows all the sinks and sources we used for our dynamic taint analysis.

Sinks	**Sources**
element.after	KeyboardEvent.charCode
element.before	KeyboardEvent.keyCode
EventSource	KeyboardEvent.key
Function.ctor	KeyboardEvent.altKey
Range.createContextualFragment(fragment)	KeyboardEvent.ctrlKey
WebSocket	KeyboardEvent.shiftKey
WebSocket.send	KeyboardEvent.metaKey
XMLHttpRequest.open(password)	KeyboardEvent.location
XMLHttpRequest.open(url)	KeyboardEvent.repeat
XMLHttpRequest.open(username)	KeyboardEvent.isComposing
XMLHttpRequest.send	MouseEvent.screenX
XMLHttpRequest.setRequestHeader(name)	MouseEvent.screenY
XMLHttpRequest.setRequestHeader(value)	MouseEvent.pageX
a.href	MouseEvent.pageY
area.href	MouseEvent.clientX
document.cookie	MouseEvent.clientY
document.writeln	MouseEvent.x
document.write	MouseEvent.y

Sinks	Sources
element.style	MouseEvent.offsetX
embed.src	MouseEvent.offsetY
eval	MouseEvent.ctrlKey
eventHandler	MouseEvent.shiftKey
fetch.body	MouseEvent.region
fetch.url	MouseEvent.movementX
form.action	MouseEvent.movementY
iframe.src	MouseScrollEvent.axis
iframe.srcdoc	PointerEvent.pointerId
img.src	PointerEvent.width
img.srcset	PointerEvent.height
innerHTML	PointerEvent.pressure
insertAdjacentHTML	PointerEvent.tangentialPressure
insertAdjacentText	PointerEvent.tiltX
localStorage.setItem	PointerEvent.tiltY
localStorage.setItem(key)	PointerEvent.twist
location.assign	PointerEvent.isPrimary
location.hash	Touch.identifier
location.host	Touch.screenX
location.href	Touch.screenY
location.pathname	Touch.clientX
location.port	Touch.clientY
location.protocol	Touch.pageX
location.replace	Touch.pageY
location.search	Touch.radiusX
media.src	Touch.radiusY
MessagePort.PostMessage	Touch.rotationAngle
navigator.sendBeacon(body)	Touch.force
navigator.sendBeacon(url)	TouchEvent.altKey
object.data	TouchEvent.metaKey
outerHTML	TouchEvent.ctrlKey
script.innerHTML	TouchEvent.shiftKey
script.src	
script.text	
script.textContent	
sessionStorage.setItem	
sessionStorage.setItem(key)	

Sinks	**Sources**
setInterval	
setTimeout	
document.source	
srcset	
track.src	
window.open	
window.postMessage	
TypedArray.setElem	
TypedArray.setFromArray	
Element.append	
BrowsingContext.PostMessage	
TextEncoder.encode	
HTMLInputElement.setValue	
Worker.PostMessage	

Table B.1: List of sinks and sources used in taint analysis to find Behavioural Biometrics scripts

Appendix C

E-Commerce

C.1 E-Commerce Websites

There are currently no publicly available website lists that consist solely of e-commerce websites. Most existing lists include a mixture of site types, such as news platforms, blogs, or informational pages where no transactions can be made. To address this gap, we curated the first comprehensive list containing only e-commerce websites. This list includes 118 websites and is presented in Table C.1.

E-Commerce Websites			
underarmour.ca	lululemon.com	choosemuse.com	rona.ca
vuoriclothing.ca	partstown.ca	epson.ca	pearsonclinical.ca
amway.ca	tuango.ca	callawaygolf.ca	oakley.com
gapcanada.ca	grainger.ca	clarius.com	travismathew.ca
bose.ca	tgod.ca	rodanandfields.com	nedco.ca
seton.ca	mosaicmfg.com	moncler.com	labaie.com
brother.ca	posterjack.ca	moroccanoil.com	garageclothing.xyz
knix.com	lovehoney.ca	hilti.ca	tinadavies.com
t-fal.ca	boathousestores.com	dynamiteclothing.com	skechers.ca
baileynelson.com	springfreetrampoline.com	spanx.com	nudestix.com
reitmans.com	bornprimitive.ca	lcbo.com	riversol.com
nationalsports.com	gnc.ca	geappliances.ca	purelyinspired.com
dogwoodbc.ca	omega.ca	velourbeauty.com	sheertex.com
herschel.com	radpowerbikes.ca	penningtons.com	indigo.ca
enfamil.ca	cubcadet.ca	rootree.ca	algosolutions.com
sheertex.com	doubleradius.com	cloudgallery.ca	thewinnerwill.com
sidekicktool.com	armament.com	sixstarpro.com	emotorsdirect.ca
gnc.ca	sportinglife.ca	cloutsnchara.com	troybilt.ca
rickis.com	irobot.ca	digitalsigntech.net	warehouseone.com
silverchef.ca	lolelife.com	bowflex.ca	coversandall.ca
cloud9canine.ca	cloud9naturally.ca	ralphlauren.ca	peel-3d.com
peace-collective.com	kcmcanada.ca	londondrugs.com	montrealweights.ca
cloudsandcoils.ca	cloudnineclothing.ca	myvega.com	saltspringcoffee.com
electrohome.com	duer.ca	westburne.ca	dufresne.ca
cloudninepajamas.com	cloutieroptical.ca	keurig.ca	nyrius.com
cloudebikes.ca	k2skates.com	twobears.ca	prohockeylife.com
blake-envelopes.com	cloudsleycellars.com	cloudbarista.com	ethique.com
blumaan.com	cdnsolar.ca	cloud9kids.ca	tdotperformance.ca
shopbeergear.ca	humbleandfume.com	hillbergandberk.com	dymon.ca
squareblockstudio.ca	thepogstore.com		

Table C.1: List of 118 curated e-commerce websites