Poster: FPFlow: Detect and Prevent Browser Fingerprinting with Dynamic Taint Analysis

Tianyi Li*, Xiaofeng Zheng†, Kaiwen Shen†, Xinhui Han*
*Peking University
{litianyi, hanxinhui}@pku.edu.cn
†Tsinghua University
{zxf19, skw17}@mails.tsinghua.edu.cn

***Abstract*—Browser fingerprint is the combination of a set of browser attributes collected by the visited website, which can uniquely identify a web user. More and more websites use browser fingerprints to track users' browsing behavior, leading to severe violation to user privacy. In this poster, we proposed FPFlow, a dynamic JavaScript taint analysis framework to detect and prevent browser fingerprinting. FPFlow monitors the whole process of browser fingerprinting, including collecting attributes, generating fingerprint, and sending it to the remote server. We evaluated FPFlow on Alexa top 10,000 websites. Our experiments showed that FPFlow could effectively detect browser fingerprinting. We found 71.3% of the websites potentially performing browser fingerprinting and revealed how browser fingerprinting is applied in real-world websites. We also showed that FPFlow could prevent browser fingerprinting with an acceptable runtime overhead without affecting JavaScript functionality.**

***Index Terms*—Browser Fingerprinting, Taint Analysis, Privacy-Enhancing Technology**

I. Introduction

Browser fingerprinting [1] is an online user tracking technique that collects browser-specific information, such as user agent, screen resolution, and installed fonts, etc., to uniquely identify the client user. By performing rendering tasks with APIs such as `Canvas` to extract hardware features, browser fingerprinting can even be used to track users across browsers.

Existing fingerprinting detection and prevention methods rely on JavaScript API access patterns or known scripts. Some "protection" methods even make the browser easier to be fingerprinted [2]. Modern browsers like Firefox have carried out countermeasures against browser fingerprinting. However, we found that Amiunique [3], a website investigating browser fingerprinting, can still uniquely identify the latest version of browsers.

Our Work. In this poster, we proposed an information flow based fingerprinting detection method. We consider a website as performing browser fingerprinting if it collects fingerprinting attributes and sends them to the remote server. We propose an in-browser dynamic taint analysis framework `FPFlow` by extending Chromium browser.

`FPFlow` marks 208 fingerprinting related attributes as taint source and 5 network-related APIs as taint sink. The V8 engine propagates taint between objects during JavaScript execution. When a request is initiated, `FPFlow` marks the request as a "fingerprinting request" if its URL or body contains a certain number of tainted data. `FPFlow` can also prevent browser fingerprinting by intercepting such fingerprinting requests.

II. Motivation

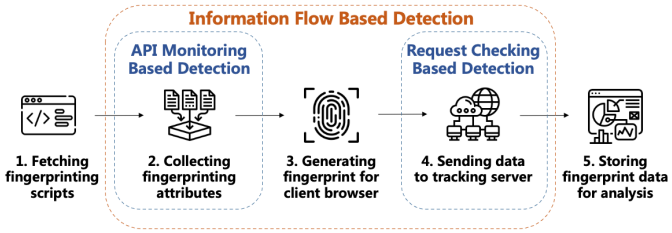

Fig. 1: The Process of Browser Fingerprinting.

Browser fingerprinting is a complex process in the JavaScript execution context. To better understand browser fingerprinting, we split the process of browser fingerprinting into five stages, as shown in Figure 1. Existing detection and prevention methods can be classified into two categories:

(1) Existing detection and prevention methods [4] against Canvas-based fingerprinting monitor the access to specific APIs on stage 2. However, they could not confirm that the rendered data is sent to the remote server, which leads to false positives.

(2) Request checking based methods [5] detect and prevent browser fingerprinting by matching fingerprinting related attributes in requests on stage 4. This method relies on known browser attributes value, so it cannot detect rendering-based fingerprinting like Canvas fingerprinting.

`FPFlow` tracks the entire life cycle of fingerprinting attributes from stage 2 to stage 4 with information flow analysis, and it can detect both browser property-based and rendering-based fingerprinting.

III. Design and Challenges

Figure 2 shows the abstract architecture of `FPFlow`. `FPFlow` extends the JavaScript engine V8 and DOM engine Blink of Chromium with taint tracking capabilities.

(1) When visiting a site, V8 first parses the JavaScript source code into AST and then generates the bytecode. `FPFlow` considers taint propagation in object property load, basic computation operations, and native function calls. `FPFlow` instruments additional bytecodes for taint propagation in bytecode generation phase.

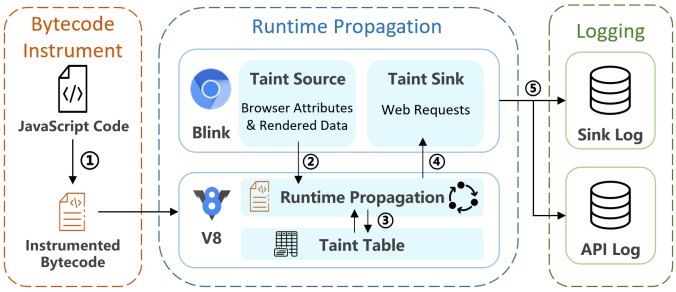

Fig. 2: Abstract Architecture of FPFlow.

(2) When JavaScript accesses the fingerprinting-related APIs in DOM, the V8 object is marked as tainted. FPFlow marks 208 fingerprinting attributes as taint source, including property-based attributes and rendering-based attributes.

(3) FPFlow maintains a taint table in V8 instance to store the taints carried by objects. During the script execution, FPFlow propagates taint between JavaScript objects and updates the taint table.

(4) When JavaScript tries to send a web request, FPFlow checks whether the request is fingerprinting request by checking whether the parameter of the request (URL, body, etc.) carries taints. If FPFlow identifies a fingerprinting request, FPFlow marks it, and can also drop it before performing the request according to user configuration.

(5) When a fingerprinting request is identified, FPFlow logs the target URL, the carried taints, the request method, and the stack trace of the request. FPFlow also logs the DOM access from JavaScript to compare with previous studies.

Besides, there are two main challenges we solved. First, FPFlow should only propagate taint for V8 native functions implemented in C++. We solve this problem by collecting the address of all native functions during the V8 bootstrap. Second, a runtime fingerprinting protection framework requires a low overhead. FPFlow reduces the overhead by optimizing the storage and propagation of JavaScript object taint.

IV. Preliminary Evaluation

We crawled the homepage of Alexa top 10,000 sites with FPFlow and analyzed the adoption of browser fingerprinting.

A. Detection of browser fingerprinting.

The experiment showed that FPFlow could detect both attributes and rendering based fingerprinting. In our experiment, we found 7,132 sites (71.3%) were **potentially** performing browser fingerprinting, and 6,654 of them sent user data to third-party domains. The most used tracking services and fingerprinting attributes are shown in table I and II.

Limitations. FPFlow uses dynamic taint analysis to analyze information flow, leading to potential false positives and false negatives. We are currently working on evaluating the accuracy of our detection result.

B. Script behavior.

Tracker loader. By analyzing the initiator of the fingerprinting requests, we found some scripts that try to load many other tracking scripts. We refer to these scripts as **tracker loader**. Different sites using tracker loader have different configurations to decide what tracking scripts to load.

TABLE I: Most Used Tracking Services

Tracker	Sites	Tracker	Sites
doubleclick.com	5,745	rubiconproject.com	1,283
google-analytics.com	4,941	adnxs.com	729
google.com	1,941	criteo.com	535
googlesyndication.com	1,672	rlcdn.com	499
facebook.com	1,556	casalemedia.com	486

TABLE II: Most Used Fingerprinting Attributes

Attribute	Sites	Attribute	Sites	Attribute	Sites
Cookie	6,829	AppVersion	4,823	CookieEnabled	3,334
UserAgent	6,827	Resolution	3,483	Language	3,310
History	4,848	Platform	3,428	Plugin	2,790

Fingerprint encoding. We found some sites encode the fingerprint attributes before sending them to the remote server, especially for Canvas fingerprint. However, we found it is hard to analyze how many sites are encoding browser fingerprint.

Fingerprinting beacon refers to many (over 5) fingerprinting requests sent to a single URL during browsing. We found 963 sites are sending fingerprinting beacons, and most of the beacon requests carry different data each time. We found function names like trackEvents and setInterval in the stack trace of these requests. We can determine by the function names that the trackers are monitoring the user's behavior at regular intervals or when a certain event is triggered.

C. Prevention of browser fingerprinting.

Our experiment shows that FPFlow can successfully block the fingerprinting requests. We manually browsed 50 sites and did not find any abnormalities. The average protection overhead of FPFlow is 9.2%.

V. Conclusion

In this poster, we propose FPFlow, a dynamic taint analysis framework to detect and prevent browser fingerprinting. FPFlow marks taint attributes as taint source and propagates taint during JavaScript execution. It checks whether web requests contain taints before they are sent out. Our preliminary result shows that FPFlow can detect and prevent browser property-based and rendering-based fingerprinting with acceptable overhead. We also revealed the behavior of fingerprinting scripts such as tracker loader and fingerprinting beacon. We are currently working on evaluating the accuracy of the detection result.

References

[1] P. Laperdrix, N. Bielova, B. Baudry, and G. Avoine, "Browser fingerprinting: A survey," *ACM Transactions on the Web (TWEB)*, vol. 14, no. 2, pp. 1–33, 2020.

[2] B. A. Azad, O. Starov, P. Laperdrix, and N. Nikiforakis, "Short paper-taming the shape shifter: Detecting anti-fingerprinting browsers," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2020, pp. 160–170.

[3] Amiunique. [Online]. Available: https://amiunique.org/fp

[4] S. Englehardt and A. Narayanan, "Online tracking: A 1-million-site measurement and analysis," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 1388–1401.

[5] N. M. Al-Fannah, W. Li, and C. J. Mitchell, "Beyond cookie monster amnesia: Real world persistent online tracking," in *International Conference on Information Security*. Springer, 2018, pp. 481–501.