BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

JSHELTER: EXTENDING INTEGRATION TESTS

ROZŠÍŘENÍ INTEGRAČNÍCH TESTŮ PROJEKTU JSHELTER

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **VLADYSLAV TVERDOKHLIB**
AUTOR PRÁCE

SUPERVISOR **Ing. LIBOR POLČÁK, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2025

Abstract

This thesis explores privacy risks from browser fingerprinting, a technique used to track users without consent for ads, profiling, and security. It examines JShelter, a browser extension that counters tracking by modifying APIs and blocking data. The study also develops a testing setup with Docker, pytest, and Selenium to evaluate JShelter's ability to protect devices, sensors, and networks.

Abstrakt

Tato práce zkoumá rizika ohrožení soukromí způsobená technikou tzv. fingerprintingu prohlížečů, která se používá ke sledování uživatelů bez jejich souhlasu za účelem zobrazování reklam, profilování a zabezpečení. Zkoumá JShelter, rozšíření prohlížeče, které proti sledování bojuje úpravou API a blokováním dat. Studie rovněž vyvíjí testovací prostředí s využitím Dockeru, pytestu a Selenium k hodnocení schopností JShelter chránit zařízení, senzory a sítě.

Keywords

jshelter, web privacy, javascript, browser extension, integration tests

Klíčová slova

jshelter, webové soukromí, javascript, rozšíření prohlížeče, integrační testy

Reference

TVERDOKHLIB, Vladyslav. *JShelter: Extending Integration Tests.* Brno, 2025. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Libor Polčák, Ph.D.

JShelter: Extending Integration Tests

Declaration

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Libora Polčáka Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Vladyslav Tverdokhlib
January 17, 2025

Contents

Chapter 1

Introduction

Web browsers are essential for accessing online content, but they can also expose users to privacy risks. Many websites misuse browser features to gather sensitive information, like user behavior, device details, and location, which can be used for tracking and profiling. This raises concerns about privacy and the misuse of personal data.

One of the main threats to privacy is fingerprinting, a technique where websites track users by identifying unique features of their browsers and devices. Fingerprinting methods are constantly improving, making it harder for users to stay anonymous. Although laws around privacy are evolving, they often do not keep up with the pace of these new tracking methods or do not attempt to do so. Additionally, while browsers have some privacy protections, they are not enough to fully prevent tracking.

To help with this, extensions like JShelter can block or modify the information that websites can collect, making it harder for them to track users. However, as fingerprinting techniques continue to evolve, it's important to regularly test these extensions to make sure they continue to provide protection.

Testing JShelter and similar extensions is crucial to ensure that they can effectively protect users from new tracking methods and continue to provide strong privacy safeguards.

Chapter 2

Understanding privacy risks and JShelter

This chapter explains the problem of web browser fingerprinting, which allows users to be tracked across the Internet. It then introduces JShelter as a solution to prevent fingerprinting and tracking. The chapter describes the key features of JShelter and focuses on how it protects sensitive information from special peripherals, such as geolocation sensors and virtual reality devices. Finally, it compares JShelter with *Google's Private Network* Access in terms of securing internal networks.

2.1 Web browser fingerprinting

Browser fingerprinting is the practice of collecting specific information to create a unique fingerprint of a user's device through their browser. Websites or applications request these data when a user interacts with them. A browser fingerprint is a set of various attributes related to the user's device, including hardware, operating system, network, and browser configuration [8]. Combining such data generates a unique identifier (fingerprint), similar to the lines and swirls on our fingertips.

The fingerprint allows websites to identify the same user across different visits or even on other sites. It is created when a user visits a website and does not require their consent. As a result, fingerprints are widely used for user tracking and, in some cases, for security purposes.

Uses of browser fingerprint

The most common use of browser fingerprints is user tracking. For example, advertising companies (for example, *Google Ads* and *Meta Pixel*) track users through fingerprints to build detailed profiles, including characteristics such as language, location, interests, and even probable age [22]. These profiles are then used for personalized advertising and marketing.

Advertising companies perform fingerprinting on their platforms and across third-party websites and domains. For example, website owners embed code from advertising services (for example, *Google Ads* and *Meta Pixel*) for monetization [1]. Furthermore, third-party data brokers can contribute by collecting user activity through fingerprints and selling it [22]. It allows user tracking to continue across different sites and sessions.

Fingerprinting is also used in banking to improve security and protect clients. For example, banks compare current fingerprints with previously stored ones for additional authentication. They can also analyze user behavior to detect suspicious activities.

Main methods of fingerprinting

Device attributes are primarily collected using JavaScript code executed when interacting with a web resource. JavaScript calls various Web APIs to extract information from the browser. In addition, network data, such as IP addresses and HTTP headers, is analyzed [24].

Key attributes include:

- Canvas and WebGL rendering
- Operating system version and screen resolution
- Language, time zone, and fonts
- Browser extensions
- Audio information
- IP addresses and HTTP headers
- Geolocation and magnetometer data

Fingerprinting is passive when it collects information from HTTP headers or network traffic. In contrast, active fingerprinting runs JavaScript code to retrieve data from browser APIs [14]. Although attributes from HTTP headers are available immediately to the fingerprinter, JavaScript calls are often used to retrieve these attributes in a detailed and precise form. One of the tools for preventing active fingerprinting is JShelter, which is the focus of this work (see Section 2.2)

The collected data are processed with a hash function to generate a unique identifier (fingerprint), a fixed-size string of letters and numbers. This makes it easier to store and compare fingerprints [24].

Unlike cookies, which store a unique ID in the browser storage for identification on websites, fingerprints are stored on the server, and users cannot delete them [8]. This makes it harder for users to avoid identification.

Fingerprint using local resources

In addition to the primary methods of fingerprinting, websites may attempt to increase its accuracy and access the user's local resources through a browser using *JavaScript*. Based on local network requests, websites can gather information about the user's applications, network nodes and their IP addresses.

One common method involves scanning for open local TCP ports. They achieve this by instructing the browser to send subsequent asynchronous HTTP requests. Browsers connect to *localhost* (*127.0.0.1*), triggering errors that indicate if a specific port is open [14].

Websites can also query local network devices, for example, printers or IoT devices, by sending requests to private IP ranges (*192.168.x.x, 10.x.x.x*) through a browser. Responses from these devices can expose details such as device type or operating system.

Additionally, websites can misuse the *Web Real-Time Communication* (*WebRTC*) protocol. *WebRTC* is a protocol that allows web browsers to exchange data in real-time, including audio and video communication. If a proxy or *Virtual Private Network* (*VPN*) is used, *WebRTC* can still reveal the user's real public IP address. It can send requests through a direct connection to the browser, bypassing the *VPN*, which may return the device's public IP address [25].

Issue of browser fingerprint

Research shows that about 90% of fingerprints are unique [6] [9]. Most websites on the Internet perform browser fingerprinting, and at least a quarter of the world's top 10,000 websites run fingerprinting scripts [3].

When visiting websites, user activity can be linked to their fingerprint. It can be used to build behavioral profiles that can be exploited for excessive personalization or to show manipulative recommendations or advertisements. In the worst-case scenario, fingerprints and tracking data could indirectly reveal a person's identity, considering details such as geolocation, browsing history, or activity patterns.

Under the *General Data Protection Regulation* (*GDPR*), fingerprinting is considered the processing of personal data. Therefore, companies must obtain explicit user consent and inform them about the purposes of using fingerprints [10]. However, in practice, compliance with these regulations remains problematic. Some companies continue fingerprinting without consent, even after users reject cookies [12].

2.2 JShelter project

JShelter is a browser extension that protects users from fingerprinting, data collection, and browser-based tracking. It creates a protective layer between users and websites. The extension detects and counters active fingerprinting that uses JavaScript (see Section 2.1) and monitors network communication between web servers and the browser [15]. JShelter primarily blocks or replaces unique data that JavaScript retrieves from the browser via APIs.

The *JavaScript API* provides endpoints that expose various types of device information (see Section 2.1). For example, JShelter introduces minor fingerprint inaccuracies or simulates sensor data, such as the device boot time [14]. It also educates users by reporting which API endpoints websites access and allows users with different expertise levels to customize protection settings.

JShelter is designed to protect users on major browsers that are available on desktop and mobile devices. It is implemented for Firefox and Chromium-based browsers such as Chrome, Opera, and Edge. This means that it can be used on devices running these browsers, including *Windows*, *macOS*, *Linux*, and *Android* devices [14].

The following subsections explain its key features: *Fingerprint Detector*, *JavaScript Shield*, and *Network Boundary Shield*.

Fingerprint Detector

JShelter uses the *Fingerprint Detector* module to detect fingerprinting. It monitors APIs that are typically relevant for creating fingerprints in real-time [16]. The detector provides

notifications and reports on the API endpoints that a webpage attempts to access, as shown in the image 2.1.

The *Fingerprint Detector* uses a heuristic approach. Instead of analyzing code, it dynamically processes and evaluates API calls. The heuristic considers two types of components: *JavaScript API* endpoints and groups of these endpoints. The *Fingerprint Detector* groups endpoints based on their semantic properties and organizes these groups into a hierarchy, assigning a weight to each endpoint and group. Thus, the detector's heuristic forms a tree, where nodes represent groups, and leaves are the endpoints with an assigned evaluation weight. Ultimately, the threat level of fingerprinting is determined according to the detection conditions [16].

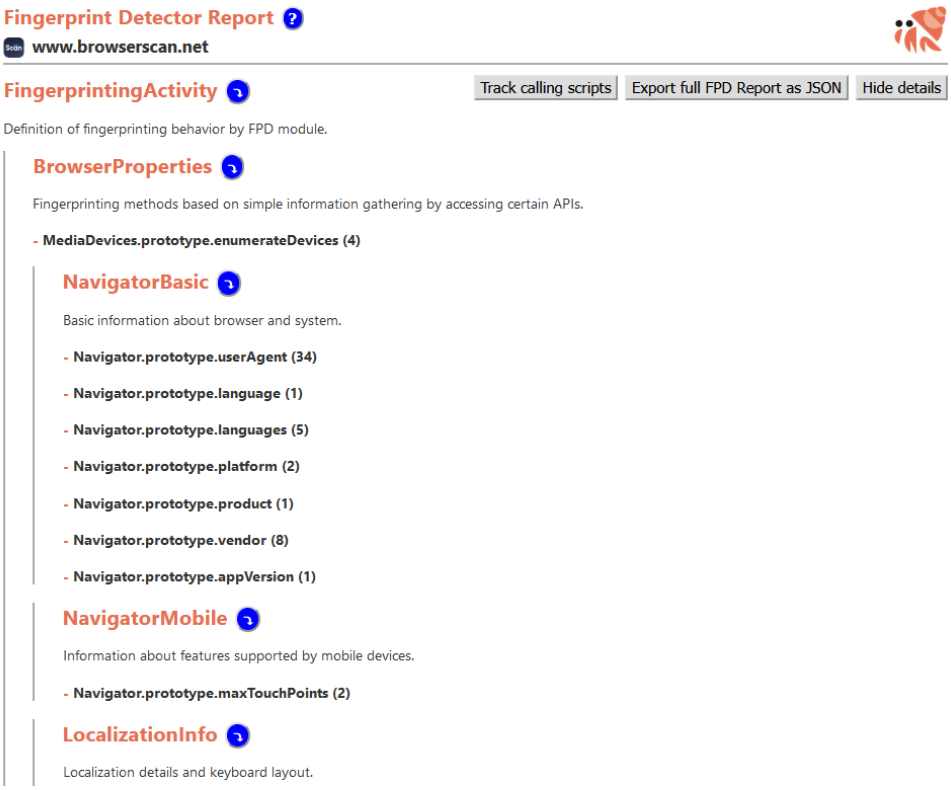

Figure 2.1: A report on API endpoints after visiting the *browserscan.net* page.

Additionally, the *Fingerprint Detector* has an optional mode to block requests from the page. In this mode, the *Fingerprint Detector* blocks all subsequent asynchronous HTTP requests from the page and clears the browser's storage where fingerprint data is stored[1].

JavaScript Shield

JavaScript Shield is the main module that modifies the browser's API to prevent fingerprinting. It blocks or modifies the results of API calls made by websites. Different values are selected for each domain and session, preventing cross-site fingerprinting. Examples of available wrappers include *HTML Canvas*, *Geolocation*, and *WebAudio*.

[1]Due to the blocking, the page may break, so the user is given the option to adjust the level of blocking.

JavaScript Shield consists of a set of wrappers. Wrappers are small snippets of code that „wrap“ the original methods or properties of the *JavaScript API* to add or change their behavior [17].

Behavior of wrappers

Wrappers are divided into three categories based on their operation [17]:

- Precision Reduction: The initial value is too precise, so JavaScript Shield reduces its accuracy.
- Provide Fake Information: Different fake data is returned for each domain and session.
- Hide Information: The initial data is unnecessary and can be hidden. In this case, an error, an empty value, or a blocked call is returned.

Protection levels

JShelter provides four main protection levels that users can choose from [17]:

- Turn *JavaScript Shield* off: Full access to the API. This option is only used for websites that the user trusts.
- Turn fingerprinting protection off: Protects only some less common APIs. It significantly reduces protection to avoid slowing down or breaking the page. Suitable when the risk of fingerprinting is low.
- Recommended: Protects main APIs against fingerprinting with recommended settings. It leaves „little lies“ to prevent cross-site fingerprinting. This level uses countermeasures that are unlikely to break the page.
- Strict: Activates full protection, including key APIs and high settings. This level blocks or returns fake values. It is designed to limit all data provided by the browser. However, it makes the user fingerprintable, as API call results are altered the same way across all domains and sessions.

Wrappers for sensors and peripherals

Sensors and peripherals refer to hardware that operates on the user's device and can be accessed through the browser. Currently, not all sensor APIs require explicit user consent, but they provide advanced physical data.

Sensors include, but are not limited to, *AmbientLight*, *Orientation*, *Accelerometer*, *Gravity*, *Gyroscope*, and *Magnetometer*. These sensors can provide detailed information about the physical environment, such as when the device was last booted [14]. They can be used for fingerprinting, as well as for eavesdropping, location tracking, and keystroke monitoring [14]. Therefore, sensor wrappers must simulate the device's physical data.

Special peripherals include *gamepads* and *virtual reality* (*VR*) devices. Such peripherals significantly expand device functionality and enhance the user experience. Since *VR* is not widespread, fingerprinting techniques can identify users through basic data, such as whether a *VR* display or game controller is connected to the computer [19]. JShelter simulates the absence of such peripherals, aligning the user with the majority.

Vulnerable timestamps

Computer clocks do not measure time accurately, and all have a unique built-in error - *clock skew*. The values can represent time with microsecond precision. Many APIs can provide timestamps.

JShelter uses rounding and randomization methods for timestamps, which is a more effective approach than simple rounding used by *Firefox Fingerprinting Protection* and *Tor Browser*. Rounding is easy to detect and can be reversed into clock skew [14].

Network Boundary Shield

The Network Boundary Shield is a module that monitors aggressive HTTP requests that a web server instructs the browser to send to the local network. In other words, If a request is sent from a public IP to a local one, the shield blocks it [18]. The primary goal is to prevent a web server from using the browser as a proxy to access the user's local network. In this way, the shield protects the user from crossing network boundaries [14].

2.3 Comparing Network Boundary Shield and Private Network Access

In this section, I will discuss the differences between *Network Boundary Shield* (*NBS*) in *JShelter* and *Private Network Access* (*PNA*) from *Google Chrome*. Both tools aim to prevent leakage of local network information through the browser, but they do so in different ways.

Private Network Access focuses on controlling access via special HTTP headers. When the browser tries to access a local resource, it sends an HTTP request with the header *Access-Control-Request-Private-Network: true*. If the local resource allows access, it responds with *Access-Control-Allow-Private-Network: true*. If the header is missing, the browser blocks the request. This method relies on the *Cross-Origin Resource Sharing* (*CORS*) protocol to control which websites can interact with local resources. *Private Network Access* protects users from *Cross-Site Request Forged* (*CSRF*) attacks that target routers and other devices on private networks. These attacks allow attackers to redirect them to malicious servers [23].

Network Boundary Shield, on the other hand, acts as a firewall between the local environment and the browser. It analyzes requests and identifies dangerous ones. It uses *IANA's* lists of IP address prefixes to detect when a request is targeting a local network. *NBS* operates through the browser's *webRequest API* and uses the DNS API to detect requests to local IPs [18]. However, because not all browsers support the *DNS API*, *NBS* has to skip the first request to obtain the IP address and then take action [14]. Unlike *PNA*, *NBS* does not rely on local resources for approval and works without their configuration.

Key differences between *Network Boundary Shield* and *Private Network Access*:

- Implementation: *PNA* controls access through HTTP headers, which requires the server to recognize and respond to the headers. *NBS* acts like a layer, blocking requests to local networks without interacting with local resources.
- User Configuration: With *PNA*, users can control access by configuring resources to allow or deny connections through headers. *NBS* gives users a more straightforward way of managing an exception list of allowed IP addresses.

- *DNS API* Support: *NBS* depends on the *DNS API* to detect local IP addresses. If the API is not supported, *NBS* allows the first request to go through and then blocks any future request. *PNA*, however, does not rely on the *DNS API* and is more consistent across browsers.

In conclusion, *Private Network Access* is simpler, as it operates with HTTP headers. However, local resources are involved in processing requests. These resources are open to HTTP communication and must be configured for *PNA* headers. *Network Boundary Shield*, on the other hand, acts as a firewall between the local environment and the browser. It is easier to set up but complex, as in the absence of the *DNS API*, it allows the request to pass through and then retrieves the IP address.

Chapter 3

Design

3.1 Integration testing environment

This section describes the preparation of the environment for integration testing. This includes creating a synthetic network, browser, server, user, and emulating devices.

Containerization with Docker

This section explains why *Docker* was chosen to create the testing environment.

Docker enables the creation of isolated and scalable containers, which is crucial for testing. Containers can create networks and emulate devices [5], such as *gamepads* or *VR* devices. It is possible to create containers for the browser with *JShelter*, a server, network, and peripherals and use *Docker Compose* to combine containers and describe their interactions [4].

Technologies and reference for the environment

The thesis of *Ms. Petráňová* [13] serves as the foundation for this project. Her work focused on developing a runtime environment for testing browser extensions. This environment automates browser extension testing for the expected protection against fingerprinting from *JShelter*. The implementation uses *Docker*, *Selenium*, and the *pytest* framework in *Python* and incorporates components from the external *PETInspector* project, which evaluates the effectiveness of security extensions.

The runtime environment developed in her thesis provides the following core functionalities:

1. Client emulation: Simulating a user visiting web pages using *Selenium* controlled by a *browser driver* for either *Mozilla Firefox* or *Google Chrome*.

2. Fingerprinting server: Implementing known methods of creating fingerprints, including HTTP requests to the local network and collecting data in *JSON* format. The server is based on *Flask*.

3. Data analysis module: Comparing the data collected by the server with the expected predefined results using *pytest*.

Building upon her work, this thesis aims to enhance and expand the existing testing environment by incorporating device emulation, requests from the server through the Synthetic network [21], and missing *JavaScript* and HTTP tests.

Specifically, the new contributions will focus on:

- Device emulation: Introducing a layer for the environment to simulate a variety of user devices and peripherals, ensuring more diverse and realistic testing scenarios.
- HTTP requesting from the fingerprinting server: Sending asynchronously and nonasynchronously *JavaScript* code with HTTP request targeting the user's local network. This will be done through the *Synthetic network* that will process all TCP flow from the web page and forward them [20] to the browser for testing *Network Boundary Shield* and *Fingerprint Detector*. The *Synthetic network* allows adjusting network conditions of *Quality of Service* (*QoS*), which creates real conditions, including instability [21].
- Extending *JavaScript* testing: Adding missing *JavaScript API* tests with peripherals to ensure better privacy evaluation of the *JShelter* extension

Peripheral emulation

Peripheral emulation is a critical addition to the integration testing environment, as it allows the simulation of various devices that interact with the browser. By emulating peripherals, we can test how the *JShelter* extension handles device-specific data and whether it provides defined privacy protection against fingerprinting methods.

Peripheral emulation in this testing environment is managed through *Docker containers*. Each *container* will be configured to emulate specific physical peripherals and connect to the browser. *Docker* allows to run devices inside the *container* by using the *–device* [5] flag. And the *uinput kernel module* [11] is used to handle these devices on *Linux*.

The integration of peripherals with the browser is achieved using *browsers API* such as *WebHID API* [7] for *Mozilla Firefox* and *chrome.hid API* [2] for *Google Chrome*, which facilitates direct communication between web pages and *Human Interface Devices* (*HIDs*). This approach allows the emulated devices to appear as real hardware.

The optimal approach for this thesis is to use physical devices at hand for peripheral emulation. *Docker* containers can encapsulate these emulated devices for integration into the testing environment and for automation purposes.

Synthetic Network

The *Synthetic Network* is a virtualized intermediate forwarder of packets that ships as a *Docker image* [21]. It is designed to simulate real-world network conditions, such as latency, bandwidth, and packet loss. This component will be integrated into the testing environment to evaluate how the *JShelter* extension interacts with network requests under controlled conditions and whether it effectively protects against web server attempts to receive a response from HTTP requests directed to the user's local network using a browser.

A *Synthetic network* is useful for testing *Network Boundary Shield* and *Fingerprint Detector*, a core features of *JShelter* blocking HTTP requests to the local network (see Section 2.2). The *Synthetic network* provides a conditioned setting to test how modules behave under realistic network conditions. It allows the emulation of unstable network flows and the adjustment of *QoS* parameters.

3.2 Integration testing and automation

This section describes the approach for extending the testing of *JShelter* using the integration testing environment described in the section 3.1 and how it can be automated. The integration testing environment based on the runtime environment by *Ms. Petráňová* and includes *containerization* with *Docker*, peripheral emulation, and requests from the server through the *Synthetic network*. The primary objective is to verify *JShelter's* effectiveness in fingerprinting protection and other privacy threats while applying new realistic scenarios, including using different devices and misusing the local network by a fingerprinting server.

Environment Setup and Container Integration

The testing environment will be built using *Docker* and *Docker Compose* to manage isolated *containers* for various components:

- Browser container derived from the *Synthetic Network* image: Includes a browser instance with *JShelter* installed and controlled by *Selenium*. In addition, it includes the forwarder *syntheticnet* that simulates realistic network conditions for pass-through HTTP requests to the browser.
- Fingerprinting server container: Runs a *Flask*-based server to perform fingerprinting and log results in *JSON*.
- Peripheral containers: Emulate devices such as a *gamepad*, *VR headset*, and their sensors.
- Data analysis module container: Compares the fingerprint data collected by the server with the expected predefined results and reports about scenarios of blocking HTTP requests.

Docker Compose is used to define and manage the interactions between these containers. The containers are interconnected, enabling controlled communication.

Figure 3.1 shows a scheme of the integration testing environment built on *Docker* that includes these components and the interactions between them.

HTTP requesting tests

The web server will send non-asynchronous and asynchronous requests of *JavaScript* code with HTTP requests targeting the user's local addresses. The traffic from the web server will be processed via the *Synthetic network*. The aim is to verify the functionality of the *Network Boundary Shield* and *Fingerprint Detector* in the following scenarios:

- Requests aimed at port scanning, such as *port 5901*.
- Requests directed to local addresses according to the list defined by *IANA*.
- Requests using domain names, such as *printer.local*.
- Requests to the addresses from a whitelist.
- Simulating unstable to test functionality under non-standard conditions.

During these tests, the fingerprinting server will save the reachability results for reports to understand if modules*JShelter* effectively block unauthorized requests while maintaining functionality for benign ones.

Wrappers tests using peripheral emulation

It includes extending the *JavaScript browser API* testing of the wrappers on the fingerprinting server. The new tests in *JavaScript* will try to retrieve new attributes related to emulated peripherals and their sensors.

Devices such as *gamepads* and *VR headsets* will be visible to the browser and tested using appropriate misuse of the *browser API*. For example, checking if a peripheral is connected using generic API such as *navigator.getGamepads()* or *navigator.activeVRDisplays()*, or reading vulnerable timestamps. The tests will also be extended to the lacking data from sensors such as *magnetometers*, *accelerometers*, and *gyroscopes*. Such tests will verify the functionality of *JShelter's* wrappers interacting with peripherals and sensors.

Enhancing the Data analysis module

The Data analysis module, built using *Python* and *pytest*, will be expanded to accommodate the new test scenarios with devices and HTTP requests. This includes:

- Comparing collected fingerprinting data against predefined expected results.
- Evaluating reports about blocking HTTP requests from the fingerprinting server.

Based on existing and added *pytest* tests for peripherals and HTTP requests, an extended evaluation of the *JShelter* will be generated.

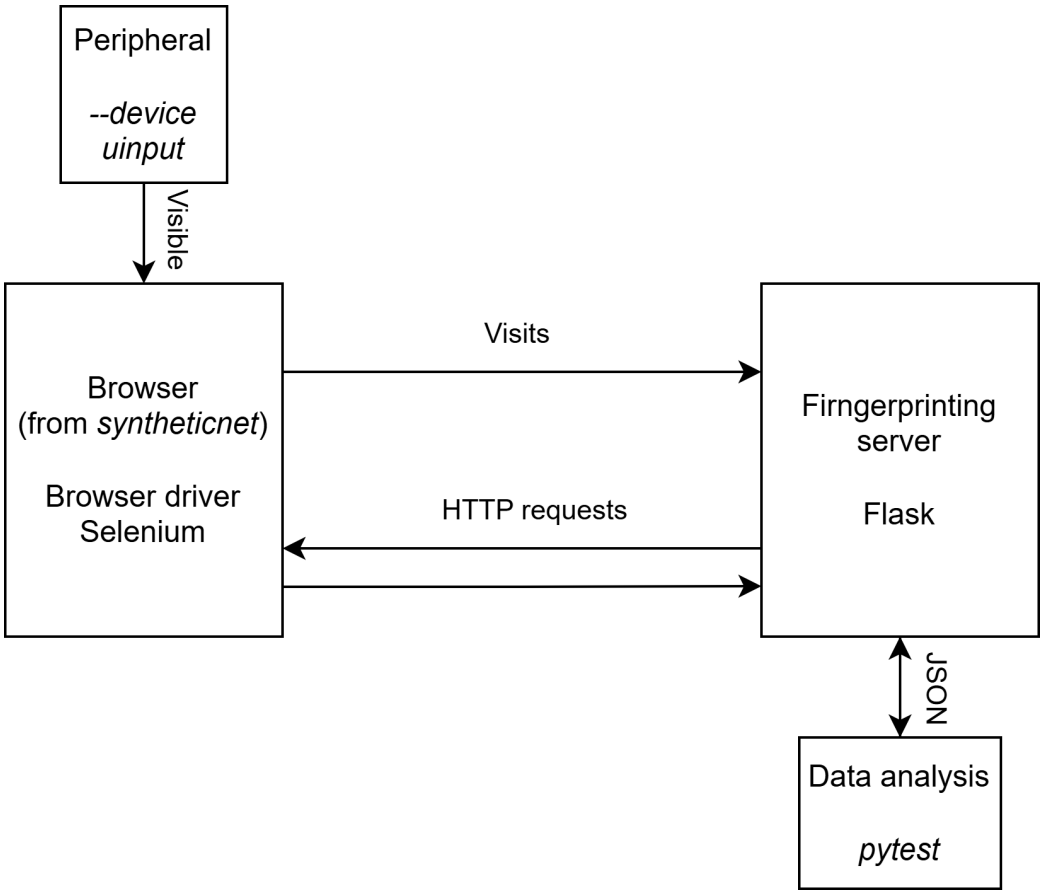

Figure 3.1: Scheme of integration testing environment built on Docker.

Bibliography

[1] Burgess, M. *The Quiet Way Advertisers Are Tracking Your Browsing* online. 26. feb 2022. Available at: https://www.wired.com/story/browser-fingerprinting-tracking-explained. [cit. 2024-11-19].

[2] Chrome, G. *Chrome.hid* online. Available at: https://developer.chrome.com/docs/apps/reference/hid. [cit. 2024-01-17].

[3] Cimpanu, C. *A quarter of the Alexa Top 10K websites are using browser fingerprinting scripts* online. 26. aug 2020. Available at: https://www.zdnet.com/article/a-quarter-of-the-alexa-top-10k-websites-are-using-browser-fingerprinting-scripts. [cit. 2024-11-18].

[4] Docker. *Docker Compose* online. Available at: https://docs.docker.com/compose/. [cit. 2024-01-16].

[5] Docker. *Running containers* online. Available at: https://docs.docker.com/engine/containers/run/. [cit. 2024-01-16].

[6] Eckersley, P. How Unique Is Your Web Browser? In: *Privacy Enhancing Technologies*. Berlin, Germany: Springer Berlin Heidelberg, Jul 2010, vol. 6205, p. 1–18. Lecture Notes in Computer Science. ISBN 978-3-642-14527-8. Available at: https://doi.org/10.1007/978-3-642-14527-8_1.

[7] Firefox, M. *WebHID API* online. Available at: https://developer.mozilla.org/en-US/docs/Web/API/WebHID_API. [cit. 2024-01-17].

[8] Laperdrix, P.; Bielova, N.; Baudry, B. and Avoine, G. Browser Fingerprinting: A Survey. *ACM Transactions on the Web*. New York, NY, United States: Association for Computing Machinery, apr 2020, vol. 14, no. 8, p. 1–33. ISSN 1559-1131. Available at: https://doi.org/10.1145/3386040.

[9] Laperdrix, P.; Rudametkin, W. and Baudry, B. Beauty and the Beast: Diverting modern web browsers to build unique browser fingerprints. In: *37th IEEE Symposium on Security and Privacy*. San Jose, United States: [b.n.], May 2016, p. 1–18. Available at: https://inria.hal.science/hal-01285470v2.

[10] Legalweb. *Browser Fingerprinting and the GDPR* online. 1. mar 2023. Available at: https://legalweb.io/en/news-en/browser-fingerprinting-and-the-gdpr. [cit. 2024-11-18].

[11] Linux. *Uinput module* online. Available at: https://kernel.org/doc/html/v4.12/input/uinput.html. [cit. 2024-01-17].

[12] Papadogiannakis, E.; Papadopoulos, P.; Kourtellis, N. and Markatos, E. P. User Tracking in the Post-cookie Era: How Websites Bypass GDPR Consent to Track Users. In: *WWW '21: Proceedings of the Web Conference 2021*. New York, NY, United States: Association for Computing Machinery, Jun 2021, p. 2130–2141. ISBN 9781450383127. Available at: https://doi.org/10.5220/0011965600003555.

[13] Petráňová, J. *Běhová prostředí pro testování činnosti rozšíření pro webový prohlížeč*. Brno, 2024. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Supervisor Ing. Libor Polčák, P. Available at: https://www.vut.cz/studenti/zav-prace/detail/155960.

[14] Polčák, L.; Saloň, M.; Maone, G.; Hranický, R. and McMahon, M. JShelter: Give Me My Browser Back. In: *Proceedings of the 20th International Conference on Security and Cryptography*. SciTePress - Science and Technology Publications, 2023, p. 287–294. ISBN 978-989-758-666-8. Available at: https://doi.org/10.5220/0011965600003555.

[15] Project, J. *About JShelter* online. Available at: https://jshelter.org/. [cit. 2024-10-25].

[16] Project, J. *Fingerprint Detector* online. Available at: https://jshelter.org/fpd/. [cit. 2024-10-26].

[17] Project, J. *JavaScript Shield* online. Available at: https://jshelter.org/levels/. [cit. 2024-11-05].

[18] Project, J. *Network Boundary Shield* online. Available at: https://jshelter.org/nbs/. [cit. 2024-11-05].

[19] Project, J. *Virtual Reality 1.1* online. Available at: https://jshelter.org/vr/. [cit. 2024-11-10].

[20] Pyne, V. and Rottenkolber, M. *Report 0* online. 21. jul 2021. Available at: https://github.com/daily-co/synthetic-network/tree/main/doc/report0. [cit. 2024-01-16].

[21] Pyne, V. and Rottenkolber, M. *Using Docker and userspace networking to simulate real-world networks* online. 23. aug 2021. Available at: https://www.daily.co/blog/using-docker-and-userspace-networking-to-simulate-real-world-networks/#introducing-synthetic-network. [cit. 2024-01-16].

[22] Rayobyte. *What Is Browser Fingerprinting?* online. Available at: https://rayobyte.com/blog/browser-fingerprinting. [cit. 2024-10-15].

[23] Rigoudy, T.; Kitamura, E. and Luo, Y. *Private Network Access: introducing preflights* online. 7. jul 2022. Available at: https://developer.chrome.com/blog/private-network-access-preflight. [cit. 2024-11-20].

[24] Seon. *What Is Browser Fingerprinting How Does It Work?* online. Available at: https://seon.io/resources/browser-fingerprinting. [cit. 2024-10-15].

[25] Vigderman, A. and Turner, G. *WebRTC Leaks: A Complete Guide* online. 26. sep 2024. Available at: https://www.security.org/vpn/webrtc-leak/. [cit. 2025-01-15].