BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

DETECTION OF SESSION REPLAY SCRIPTS

ROZPOZNÁNÍ PŘÍTOMNOSTI SKRIPTŮ PRO ZÁZNAM SEZENÍ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. MARTIN VOŠČINÁR**
AUTOR PRÁCE

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

BRNO 2025

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Master's Thesis Assignment

162591

Institut: Department of Information Systems (DIFS)
Student: **Voščinár Martin, Bc.**
Programme: Information Technology and Artificial Intelligence
Specialization: Cybersecurity
Title: **Detection of Session Replay Scripts**
Category: Web
Academic year: 2024/25

Assignment:

1. Study the problem of session recording scripts in a web browser. Find session recording services that offer free testing plans, or find pages containing these scripts.
2. Get familiar with browser extension development and study the architecture and implementation of JShelter.
3. Propose a methodology describing the behaviour of session recording scripts and discuss the methodology with the supervisor. Focus on script behaviour so that the methodology can detect session recording scripts independently on the domain where the script is located.
4. Design a tool that detects session recording scripts, for example as a new JShelter shield, or as a new browser extension. Ask the supervisor for approval.
5. Implement the proposed tool.
6. Test the implementation. If you decide to extend JShelter, consult the testing and its results with the JShelter project team.
7. Evaluate the work and suggest possible future directions of the developed tools.

Literature:

- Acar, G., Englehardt, S., and Narayanan, A. (2020). No boundaries: data exfiltration by third parties embedded on web pages. Proceedings on Privacy Enhancing Technologies, 2020, pages 220–238
- Grodzinsky, F. S., Miller, K. W., and Wolf, M. J. (2022). Session replay scripts: A privacy analysis. The Information Society, 38(4), pages 257–268.
- Polčák, L.; Saloň, M.; Maone, G.; Hranický, R. and McMahon, M. (2023). JShelter: Give Me My Browser Back. In Proceedings of the 20th International Conference on Security and Cryptography, ISBN 978-989-758-666-8, ISSN 2184-7711, pages 287-294.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor: **Polčák Libor, Ing., Ph.D.**
Head of Department: Kolář Dušan, doc. Dr. Ing.
Beginning of work: 1.11.2024
Submission deadline: 21.5.2025
Approval date: 21.10.2024

Abstract

This thesis focuses on developing a method for detecting session replay scripts in a browser extension. It summarizes the current state of research on the topic of session replay scripts along with their privacy implications, and illustrates possibilities of browser extensions. It tries to propose a methodology for describing session replay script behavior for the implementation of detection methods.

Abstrakt

Táto práca sa zameriava na návrh a vývoj metódy detekcie skriptov na záznam sedenia v rámci rozšírenia pre prehliadač. Zhŕňa súčasný stav výskumu k týmto skriptom a ich dopadov na súkromie, a ukazuje možnosti tvorby rozšírení prehliadača. Pokúša sa navrhnúť metodológiu popisu správania skriptov pre záznam sedenia, ktorá má byť použitá pre implementáciu metód detekcie.

Keywords

session replay, privacy, web, browser extensions, JavaScript

Klíčová slova

záznam sedenia, súkromie, web, rozšírenia prehliadača, JavaScript

Reference

VOŠČINÁR, Martin. *Detection of Session Replay Scripts.* Brno, 2025. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Libor Polčák, Ph.D.

Detection of Session Replay Scripts

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Ing. Libor Polčák, Ph.D.. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

. . . . . .
Martin Voščinár
21.5.2025

Acknowledgements

I want to thank my thesis supervisor, Ing. Libor Polčák, Ph.D., for his time, patience, and valuable advice. My endless gratitude extends to my family for their unwavering support and encouragement throughout my journey.

Contents

1 Introduction **2**

2 Session recording scripts **3**
2.1 Session replay services 3
2.2 Impact on privacy 5
2.3 Session recording in practice 5

3 Browser extensions **8**
3.1 Basics of browser extensions 8
3.2 JShelter 11

4 Detection methods **14**
4.1 Behavior of session recording scripts 14

5 Conclusion **19**

Bibliography **20**

Chapter 1

Introduction

Data has become one of the most valuable assets in today's interconnected world. Every action we take online – whether browsing a website, using social media, or shopping generates data, which is often collected by multiple parties. These include the websites we browse, third-party advertisers, analytics companies, or data brokers, which monetize data by selling to other third parties. While some of the data is being used for our benefit, much is being collected without our consent and used for purposes we are oblivious to. Thus, the extent of data collection has brought significant risks to individual privacy.

In order to take steps to protect our privacy, we must educate ourselves on what data is being collected from us and how it is used to shed light on unwanted practices. This thesis aims to do just that by improving the detection of session replay scripts. Session replay scripts record the user's interactions with a website, including mouse movements, clicks, and key presses, which are then sent to the service provider. This data is then used by website owners to improve the user experience. However, previous research [3] suggests these scripts can exfiltrate personal data if implemented incorrectly. Methods used by researchers to detect these scripts are not practical for ordinary people. Therefore, the goal of this thesis is to expand the functionality of an existing web extension – JShelter, to include the detection of session replay scripts.

Chapter 2 introduces session replay services, the companies offering them, the session recording scripts they utilize, and their impact on privacy. Chapter 3 describes the components of browser extensions and the architecture of JShelter. Chapter 4 proposes a method of session replay script detection to be expanded upon in the planned browser extension module implementation.

In the current unfinished state, the thesis only contains research and a few experiments that were done as a part of the term project.

Chapter 2

Session recording scripts

Session replay is a service offered by some web analytics companies. For a web application developer, it requires incorporating supplied session recording scripts, also called session replay scripts, into the web application. These scripts record the user's interactions with the website, including mouse movements, clicks, and key presses, which are then sent to the analytics provider. Depending on the provider, recorded interactions can be utilized in a number of ways, e.g., a literal replay of user behavior or, more likely, aggregated and visualized as heat maps or scroll maps. While this can be beneficial to web developers and website owners in improving the user experience, it also raises privacy concerns, as the data can often be collected without the user's knowledge and can potentially contain sensitive user information [3, 5, 7].

This chapter introduces session replay scripts and marketing claims of the providers offering these services. Previous research is presented, on the subject of session replay scripts' impact on privacy, their detection, and usage in practice.

2.1 Session replay services

Session replay services are marketed primarily as tools that enhance user experience and optimize website performance by providing insights into user behavior [5]. Companies offering these services emphasize several key benefits, including a better understanding of user behavior, conversion optimization, and customer support enhancement. For marketing and UI/UX teams, companies highlight their potential to understand which elements grab the users' attention and to improve conversion rates on online shops, meaning to optimize the number of users that complete a purchase and reduce drop-outs by identifying where the users get lost [1, 2]. Another selling point is enhanced customer support – session replay services are promoted as tools that empower support teams to resolve customer issues more effectively. Using these tools, customer support employees are able to understand users' complaints better [1]. The session replay providers also like to show that they preserve users' privacy, offering options to opt out from data collecting and mask user inputs [8].

As previously mentioned, the session recording can be utilized for playback, with highlighted user interactions which can then be analyzed. Other visualizations typically show aggregated data. One example are heat maps – these can use color intensity to visualize the number of user click in a certain area. Alternatively, they can show mouse movements, which would highlight areas the user hovers on with his mouse as can be seen on Figure 2.1.

User behavior can also be analyzed while scrolling, to identify how far down the users scroll on the page, as seen on 2.2.

Figure 2.1: Mouse-tracking heat map [10]

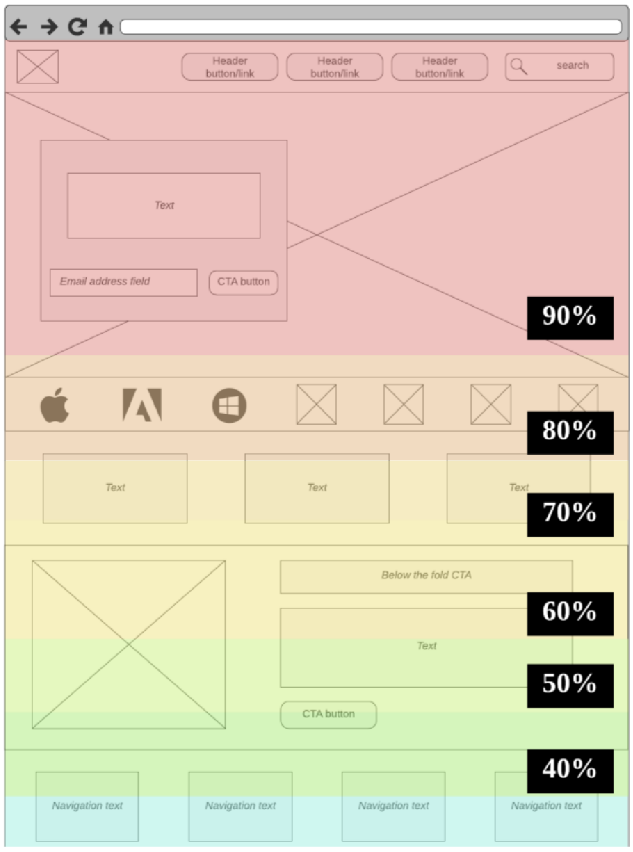


Figure 2.2: Scroll map [10]

2.2 Impact on privacy

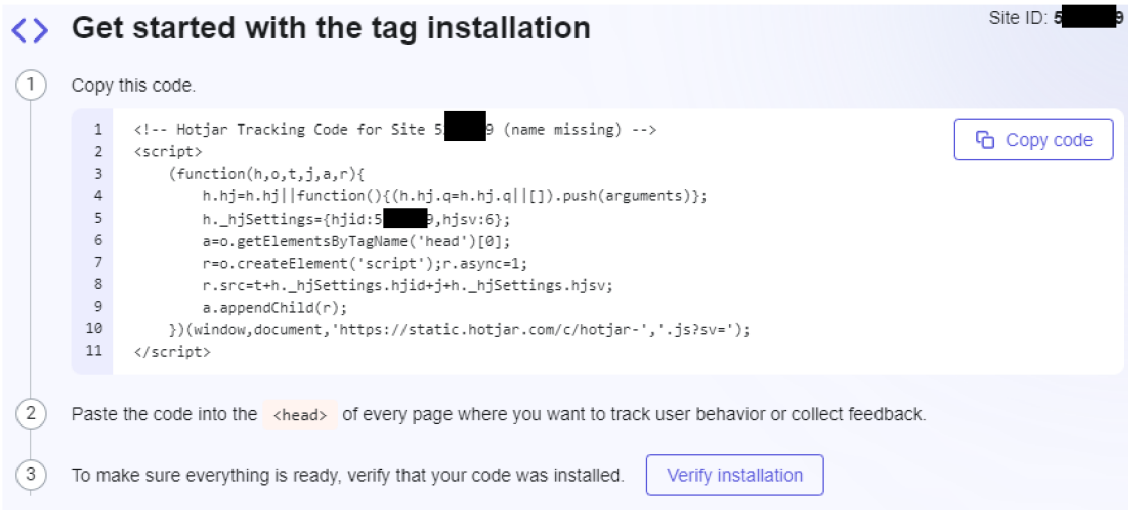
Session replay services utilize session replay scripts embedded into the web page to record the user interaction. These scripts have the same access to the Document Object Model (DOM) content as the user. They record user behavior by essentially monitoring changes in the DOM and different mouse, keyboard, or touch panel events. The whole DOM content can also be exfiltrated this way, along with everything that is displayed on the web page [3]. This raises significant privacy concerns. The DOM of the web page can contain sensitive information, such as the person's name, e-mail or physical address, phone number, ID number, banking details, medical or travel history, and much more. Even recorded keypress events can theoretically help uncover the user's left- or right-handedness, which could be considered sensitive information, too. In short, session replay scripts can lead to Personally Identifiable Information (PII) leakage if incorrectly implemented [3].

This does not have to be done maliciously, and in the case of session replay scripts, it is entirely unlikely to be. Plus, the session replay service providers offer countermeasures in the form of redaction tools. They include automated redaction features, which differ between the companies. Some prefer to mask data with a string of equivalent length, while others exclude it altogether. The differences also lie in the type of data that is automatically redacted. In 2017, none of the companies tested by Acar, G. et al. [3] automatically redacted displayed DOM content, only the user input forms, although some offered options for manual redaction. Automatic redaction was done by input element type or heuristics, which may not always match the implementation by web developers. Furthermore, only two – UserReplay and SessionCam masked or excluded all inputs by default, with the option of a whitelist. Others opted for developers to blacklist unsafe input forms which were not automatically detected, an inherently more dangerous solution. Because of this, multiple session replay companies allowed password, credit card, and health information leaks [4]. At that time, some companies even allowed recordings to be explicitly linked to the user's real identity. In total, session replay scripts were discovered on 482 out of the top 50 000 sites according to Alexa. The research was done back in 2017, so the features offered by companies have changed, and more companies have started to offer automatic redaction of user inputs [7] [8] [10].

Ethics of this type of data collection were called into question by Grodzinsky et al. [7]. This is because the users are typically unaware they are being monitored; even website owners and developers may not be completely aware. Researchers even suggest potential violations of existing privacy laws, such as GDPR in the EU, HIPAA, and FERPA in the USA [3]. Users can somewhat protect themselves by using ad-blocking extensions, which include blocklists such as EasyList/EasyPrivacy [3]. This is until companies change session replay script locations, their whole domains, or new such companies form and blocklists need to be updated.

2.3 Session recording in practice

Session recording scripts can be typically installed in multiple ways, depending on the technology powering the website. For example, Hotjar offers installation through npm or onto different platforms, like Google Tag Manager and WordPress. Most session replay companies offer installation using an HTML snippet with a few lines of JavaScript code inside a `<script>` tag 2.3. This way, they also market their service as being very easy to use.



Get started with the tag installation

Site ID: 5[redacted]9

1 Copy this code.

Copy code

```
<!-- Hotjar Tracking Code for Site 5[redacted]9 (name missing) -->
<script>
    (function(h,o,t,j,a,r){
        h.hj=h.hj||function(){(h.hj.q=h.hj.q||[]).push(arguments)};
        h._hjSettings={hjid:5[redacted]9,hjsv:6};
        a=o.getElementsByTagName('head')[0];
        r=o.createElement('script');r.async=1;
        r.src=t+h._hjSettings.hjid+j+h._hjSettings.hjsv;
        a.appendChild(r);
    })(window,document,'https://static.hotjar.com/c/hotjar-','.js?sv=');
</script>
```

2 Paste the code into the `<head>` of every page where you want to track user behavior or collect feedback.

3 To make sure everything is ready, verify that your code was installed. Verify installation

Figure 2.3: Hotjar HTML code snippet

Many session replay companies offer free trials. For the purpose of studying the scripts, we can create accounts and install them one at a time on a dummy webpage. The list of session replay providers who offer trial periods contains:

- Amplitude – https://amplitude.com/,
- Datadog – https://www.datadoghq.com/,
- Fullstory – https://www.fullstory.com/,
- Heap – https://www.heap.io/,
- Hotjar – https://www.hotjar.com/,
- LogRocket – https://logrocket.com/,
- Lucky Orange – https://www.luckyorange.com/,
- OpenReplay – https://openreplay.com/,
- Posthog – https://posthog.com/,
- Sentry – https://sentry.io/,
- Smartlook – https://www.smartlook.com/,
- Userpilot – https://userpilot.com/.

These services can be used for experimentation and analysis of script behavior, leading into methods of detection, later in the chapter 4. Acar used a two-step approach for detection, but they aimed to detect services collecting page content, not only session replay scripts [4]. In short, they inserted a distinctive value into the webpage's HTML and looked for signs of that value being transmitted to a third party in the page's traffic, even if it was encoded or hashed. Then, they focused on pages where a significant amount of data was collected during the visit, but for which they did not find a unique ID. On these websites, they injected a data chunk into the page and monitored whether they noticed a

corresponding increase in the quantity of data sent to the third party. This way, they found at least 482 websites, as was previously mentioned. Additionally, some of the web pages found still use scripts by session replay providers, albeit different ones. These include at least `redhat.com`, using Hotjar[1], and `sky.com`, `autodesk.com` using Dynatrace[2].

[1] https://www.hotjar.com/
[2] https://www.dynatrace.com/platform/session-replay/

Chapter 3

Browser extensions

Browser extensions are essentially software modules that enhance or modify the functionality of a web browser. They allow users to customize their browsing experience by modifying web pages, adding features, automating tasks, or integrating with external services. Extensions can perform various functions, from blocking ads and managing passwords to providing custom themes. They operate by interacting with a browser's APIs and the content of web pages. One such extension in JShelter is introduced later in this chapter.

3.1 Basics of browser extensions

Browser extension utilize WebExtensions APIs, some of which include [6]:

- **Events API** used to add and manage event listeners,
- **Permissions API** allows you to request new permissions and inspect current ones,
- **Messaging API** used to send messages between parts of the extension, between the extension and the web page, another extension or an external program,
- **Storage API** allows you to manage a key-value storage,
- **Identity API** used for authentication,
- **Network requests APIs**, which include webRequest API, webNavigation API and declarativeNetRequest API,
- **Action API** to control the extension toolbar icon and its click handlers,
- **Cookies API** is an interface to manage cookies,
- **Bookmarks API** is an interface to manage bookmarks,
- **History API** is an interface to manage browsing history,
- **Tabs and windows APIs** used to create, remove, inspect, modify, rearrange, pin, and mute tabs and windows,
- **Alarms API** enables extensions to schedule code to run in the future,
- **Scripting API** used to programmatically inject JavaScript or CSS into the page,

Browser extensions are made up of several components, most of which are optional [6]:

- **Manifest file** is essential to any extension because it serves as its core configuration file. It specifies metadata about the extension in JSON format, including its name, author, version, and description. It also lists paths of source files used in the extension, assigned to the appropriate key, such as: `'options_page': 'options.html'`.

- **Background script(s)** primarily handle browser events, such as browser actions or messages from other parts of the extension. They can be implemented as one or more JavaScript files or a single HTML file, which includes scripts using a `<script>` tag. Background scripts can exchange messages and interact with browser APIs, such as those for bookmarks, tabs, and storage, or insert scripts into a website. They can be persistent and run in memory for the whole browser runtime or nonpersistent and only run on demand when an event occurs.

- **Content script(s)** can modify web pages using JavaScript and CSS. The injected JavaScript code can then modify the page DOM and send network requests but runs sandboxed from the page runtime. They also have limited access to browser APIs and can exchange messages with other parts of the extension.

- **Popup page** is used to display a custom user interface inside the native interface of a browser 3.2. It appears in a dialog window after clicking the extension's toolbar icon. Typically, it is used for some frequently used functionality. The page is freshly rendered each time it is opened and has the same capabilities as background scripts.

- **Options page** is another possibility for displaying a custom user interface. It is rendered as a standalone web page that is opened through the extension context menu or details page. It is just as capable as the popup page but is used for less frequent functionality.

- **Devtools page** is a similar option, but it is displayed as a panel or sidebar in devtools.

Of course, depending on the extension, not all of these are necessary. Other files can also be included, such as icons or accessible resource files. These components can be visualized in a diagram, as can be seen in Figure 3.1.

Each webpage is rendered in the user interface of a browser and has its own JavaScript runtime. However, webpages in multiple tabs are sandboxed and have their own JavaScript environment, unless they have matching origins – in which case, they are able to share some resources. Web extensions are similar to web pages in browser tabs in the sense that they are also sandboxed entities with their own JavaScript runtime and typically also a HTML user interface. But on top of that, extensions have access to browser interfaces, context menus, and a separate set of APIs, which are the aforementioned WebExtensions [6].

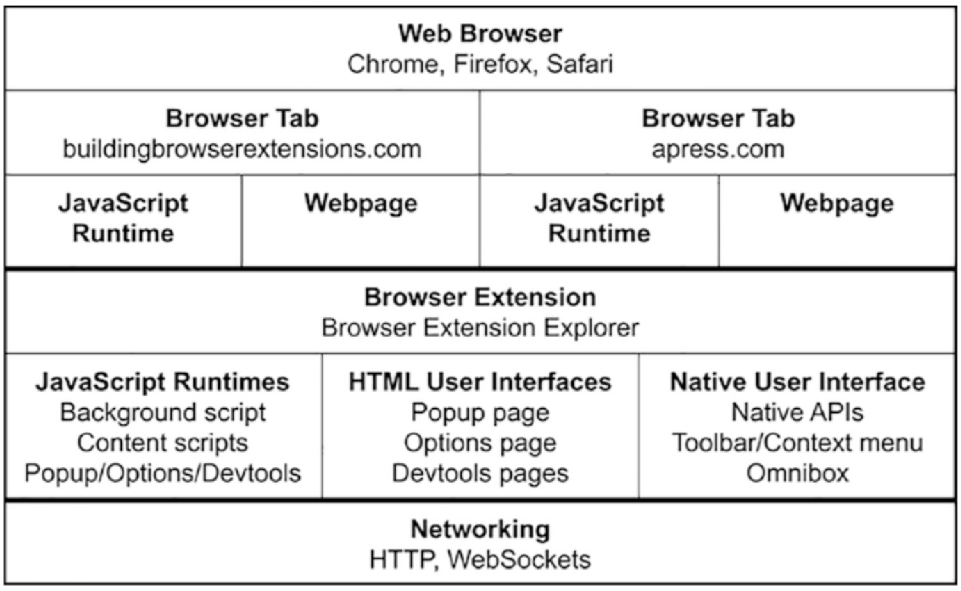


Figure 3.1: Diagram of a web browser with multiple tabs and an extension installed [6]

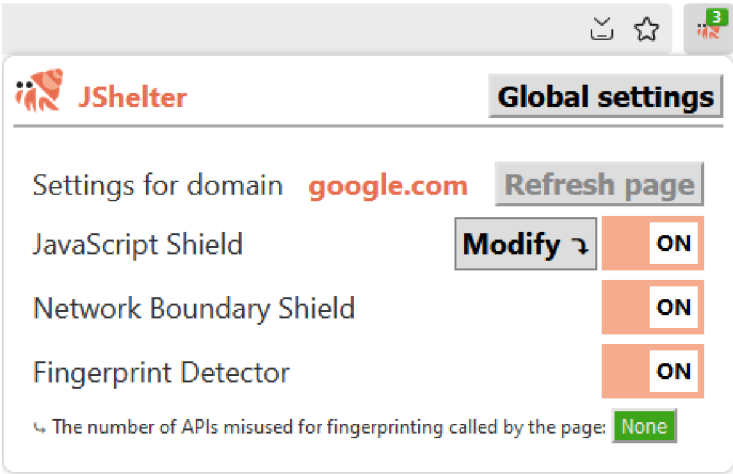


Figure 3.2: JShelter extension popup page example

Nowadays, the most prevalent browsers are based on Chromium, an open-source browser project, which means they share mostly the same APIs. These include Google Chrome, Microsoft Edge, Brave, Opera, and Vivaldi as the most popular ones. Firefox and Safari have also adapted to support most of the same Web Extensions API to make web extension development easier. The main company maintaining Chromium, Google, obviously has a great influence and is currently using it in a push to change the way extensions are implemented to restrict their capabilities. This change is accompanied by a definition of a new `manifest_version 3` in the `manifest.json` file instead of 2. Therefore, these changes are commonly referred to as Manifest V3 [6].

Aside from the security and performance motivations behind this change, manifest V3 brings major changes in two areas: background scripts and webRequest API. Background scripts are replaced by service workers, which act similarly to a nonpersistent background script. They are no longer implemented as a headless page and can only be run on demand.

This means that special Alarms API must be used to schedule code to run in the future, not `setTimeout()` or `setInterval()`. The webRequest API is being replaced by DeclarativeNetRequest, which only allows extensions to supply a static set of rules for intercepting traffic that are then executed by the browser. This is opposed to webRequest API, which lets developers intercept network requests and run JavaScript code at different points in the request lifecycle. All in all, these changes will limit the capabilities of extensions designed for ad blocking and tracking prevention [6].

3.2 JShelter

JShelter is an open-source web extension designed to enhance user privacy and security by mitigating various forms of online tracking and fingerprinting. As described by Polčák et al. [9], it allows users to tweak the browser APIs to prevent unauthorized access to sensitive user information and device characteristics. JShelter employs a heuristic approach to detect and block fingerprinting behavior. Additionally, it creates a report that explains detected fingerprinting APIs to the user. Attempts to misuse the browser as a proxy to access the local network are also blocked. This project incorporates previous research in little-lies-based fingerprinting prevention and is inspired by techniques used by the unmaintained Chrome Zero [11] and Web API Manager [12] projects. It uses an open-source NoScript Commons Library[1], which can be used, e.g., to reliably inject JavaScript code into the JavaScript context of a web page before the page scripts can access the context.

JShelter offers three types of protection:

- **Fingerprint Detector (FPD)**, which monitors APIs that are frequently used for fingerprinting and applies heuristic analysis to detect fingerprinting behavior in real-time. FPD notifies the user upon detection of such attempts and also allows the user to configure blocking subsequent requests by the fingerprinting page. Additionally, the user has the freedom to choose the aggressiveness of this approach, which could break the page behavior. FPD also offers a report that summarizes its findings from the visited web page to inform users about fingerprinting and explain the reasons behind FPD's notification or possible blocking.

- **JavaScript Shield (JSS)**, which modifies or disables JavaScript APIs commonly used for fingerprinting. It focuses on modifications and spoofing of created fingerprints, which can be configured by the user to different levels. range from slightly modifying the results of API calls on different domains to using completely fake values and blocking the API completely. JSS utilizes farbling by modifying values accessed by page scripts with different little lies to mitigate the possibility of cross-site tracking. Additionally, readings of many sensors, such as orientation, accelerometer, geolocation, and more, are modified to use artificially generated fluctuating values.

- **Network Boundary Shield (NBS)** monitors the source and destination of web requests to detect attempts to use the browser as a proxy to the local network. This functionality is only fully supported in Firefox.

[1] https://github.com/hackademix/nscl

JShelter code repository consists of the following source file structure:

```
jshelter/
├── chrome/
│   ├── http_shield_chrome.js
│   ├── manifest.json
│   ├── manifest.json.license
│   └── service_worker.js
├── common/
│   ├── background.js
│   ├── common.css
│   ├── code_builders.js
│   ├── document_start.js
│   ├── http_shield_common.js
│   ├── levels.js
│   ├── fp_*.js
│   ├── wrapping*.js
│   ├── fp_report.{js, css, html}
│   ├── options_*.{js, css, html}
│   ├── popup_*.{js, css, html}
│   └── ...many other folders and files...
├── firefox/
│   ├── http_shield_firefox.js
│   ├── levels_browser.js
│   ├── manifest.json
│   ├── manifest.json.license
│   └── options_nbs.js
├── nscl/
│   └── ...NoScript Commons Library source files..
├── tests/
│   └── ...benchmark, integration, performance, system and unit tests...
├── wasm/
│   └── ...implementation of farbling in WebAssembly...
├── website/
│   └── ...source files of the JShelter website...
├── Makefile
└── ...other folders and files...
```

As we can see, the code for Chrome and Firefox extensions contains separate `manifest.json` files and a few scripts, but most of the code is shared. The Chrome version uses Manifest V3, so it must use a `service_worker.js` script, which in turn imports the appropriate scripts in `common/`. On the other hand, Firefox has some additional options not available in Chrome. Apart from this, most of the functionality is contained inside `common/`.

`document_start.js` is the main script that is launched when the browser loads a page and `background.js` is the main background script. As the name suggests, FPD is implemented in `fp_*.js` scripts; NBS is implemented in `http_shield_{chrome, firefox, common}.js` scripts. Popup, options, and FPD report pages also have their own implementation inside the folder. JSS consists of a lot of scripts, including `wrapping*.js` and

`code_builders.js`. These contain method wrappers for various APIs, in addition to code required to join and inject the wrappers and scripts into the web page.

Chapter 4

Detection methods

In order to detect session replay scripts, we need to understand their behavior first. A simple way to start is to use DevTools, which are offered by all of the most used browsers. For this purpose, we will use DevTools offered in Chromium browsers. Subsequently, this chapter will contain proposed detection methods of session replay scripts.

4.1 Behavior of session recording scripts

The websites mentioned in section 2.3 can be used to observe the script behavior. Session replay scripts typically utilize event listeners and communication with the server that provides them. To document this behavior, native JavaScript functions can be wrapped by our implementation, as illustrated in listing 1:

```
const originalAddEventListener = EventTarget.prototype.addEventListener;
EventTarget.prototype.addEventListener = (type, listener, options) => {
    console.log('Event listener added:', {this, type, listener, options});
    originalAddEventListener.call(this, type, listener, options);
};
const originalRemoveEventListener =
↪   EventTarget.prototype.removeEventListener;
EventTarget.prototype.removeEventListener = (type, listener, options) => {
    console.log('Event listener removed:',{this, type, listener, options});
    originalRemoveEventListener.call(this, type, listener, options);
};
const originalFetch = window.fetch;
window.fetch = function (...args) {
    console.log('Fetch call intercepted:', args);
    return originalFetch.apply(this, args);
};
const originalXHR = XMLHttpRequest.prototype.open;
XMLHttpRequest.prototype.open = function (...args) {
    console.log('XHR call intercepted:', args);
    return originalXHR.apply(this, args);
};
```

Listing 1: Simple JS function wrappers

In order to avoid logging the function calls from other scripts, the wrapper should be utilized only for the duration of session replay script execution.

To test this, two mostly static websites were set up, one with embedded Hotjar and the other with Lucky Orange session replay scripts. A breakpoint was set on the first line of the script, where execution should start once it is loaded using the code snippet in the page HTML. In both cases, many `addEventListener` calls were logged. Interestingly, some events had multiple event listeners added. Specifically, Hotjar and Lucky Orange, respectively, added listeners to these events:

```
document: click, scroll, mousedown, mousemove, keypress, cut, copy, paste,
↪  input, blur, visibilitychange, change
Window: pagehide, beforeunload, scroll, resize, error, unhandledrejection

document: click, scroll, mousedown, mouseup, mousemove, keydown, keyup,
↪  input, submit, blur, visibilitychange, change, focus, animationstart,
↪  transitionstart, startTracking
Window: click, mousemove, scroll, resize, error, message, hashchange
```

Listing 2: Logged listeners added for Hotjar and Lucky Orange

Hence, there are some common events, some similar and some completely different. In the case of Lucky Orange, additional event listeners were being added to the same events periodically after some time:

```
document: blur, click, focus, submit, input, mousemove, scroll, resize
Window: resize
```

Listing 3: Listeners repeatedly added by Lucky Orange

Both Hotjar and Lucky Orange load the first script using the snippet in HTML, which seems to initialize the settings and load configuration specific to the session replay service user, then load another script that provides the actual session recording. In the case of Hotjar, seemingly all of the necessary configuration is included in the first script, as there is no communication with the server at the start. Lucky Orange, on the other hand, uses `fetch()` to load some configuration in the first script. In both cases, as can be seen in Figures 4.1 and 4.2, data is being sent to the server after some time has passed.

Name	Status	Type	Initiator	Size
ws?v=7&site_id=5[redacted]0	101	websocket	modules.1f	0 B
?site_id=52[redacted]0&gzip=1	200	xhr	modules.1f	170 B
hotjar-5[redacted]0.js?sv=6	304	script	index.html:	401 B
modules.1f[redacted]4ab...	200	script	hotjar-5267	(memory cache)

Figure 4.1: Hotjar communication with the server

Name	Status	Type	Initiator	Size
socket.io/?EIO=3&transport=websocket	101	websocket	core.js?v=49	0 B
public-auth	200	fetch	core.js?v=49	14 B
events:publish	200	fetch	core.js?v=49	88 B
recording-data:publish	200	fetch	core.js?v=49	88 B
recording-data:publish	200	fetch	core.js?v=49	88 B
events:publish	200	fetch	core.js?v=49	88 B
recording-data:publish	200	fetch	core.js?v=49	88 B
socket.io/?EIO=3&transport=websocket	101	websocket	core.js?v=49	0 B
socket.io/?EIO=3&transport=websocket	101	websocket	core.js?v=49	0 B
socket.io/?EIO=3&transport=websocket	101	websocket	core.js?v=49	0 B
socket.io/?EIO=3&transport=websocket	101	websocket	core.js?v=49	0 B
mqtt	101	websocket	core.js?v=49	0 B

Figure 4.2: Lucky Orange communication with the server

Hotjar and Lucky Orange were selected as one example of the more known and one less known provider of session replay services in this exploration. Another big provider is Dynatrace. One example of a website using services of this provider is `autodesk.com`.

While on this website, if we look at the event listeners in DevTools of any Chromium browser, we see that, for example, under `DOMContentLoaded`, there is an event handler with a function from a script hosted on the domain `dynatrace.com` 4.3.

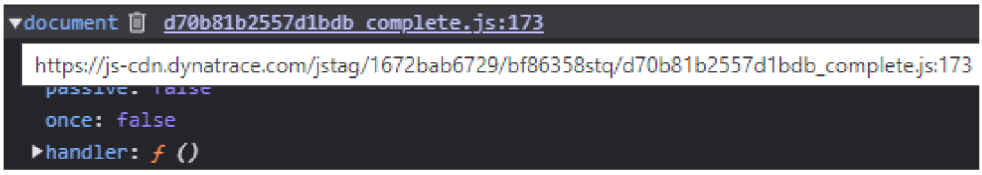


Figure 4.3: One of `DOMContentLoaded` event handlers on `autodesk.com`

If we step through the code in the debugger, we will see that it initializes its settings and cookie value, communicates with its server using `XMLHttpRequest`, and eventually sets up event listeners. The events it adds listeners to are:

```
document: click, dblclick, scroll, mousedown, mouseup, keydown, keyup,
↪  touchstart, touchend, readystatechange, visibilitychange, change,
↪  DOMContentLoaded
Window: test, pageshow, pagehide, beforeunload, popstate, load, hashchange,
↪  unhandledrejection
Performance: resourcetimingbufferfull
```

Listing 4: Listeners added by Dynatrace

In the script, there is code that looks like some DOM API methods are being overridden. To confirm this, overridden methods can sometimes be detected by looking at the string representation of a function, whether it contains `'[native code]'` as seen in 5.

However, the possibility of the function being overridden cannot be dismissed this way, as the code could deliberately contain that string to avoid this primitive detection.

```
function isOverridden(object, methodName) {
    const method = object[methodName];
    if (!method) return false; // Method does not exist
    return method.toString().indexOf('[native code]') === -1;
}
```

Listing 5: Code for detection of an overriden method

This way, we can focus on the functions that could most likely be used in session replay scripts, such as event listener mechanisms, DOM editing or `XMLHttpRequest` and `fetch` calls for communication with the server:

```
const methods = [
    { object: EventTarget.prototype, method: 'addEventListener' },
    { object: EventTarget.prototype, method: 'removeEventListener' },
    { object: EventTarget.prototype, method: 'dispatchEvent' },
    { object: MutationObserver.prototype, method: 'observe' },
    { object: Node.prototype, method: 'appendChild' },
    { object: Node.prototype, method: 'removeChild' },
    { object: Node.prototype, method: 'insertBefore' },
    { object: document, method: 'createElement' },
    { object: document, method: 'querySelector' },
    { object: document, method: 'querySelectorAll' },
    { object: XMLHttpRequest.prototype, method: 'open' },
    { object: XMLHttpRequest.prototype, method: 'send' },
    { object: window, method: 'fetch' },
    { object: window, method: 'clearTimeout' },
    { object: window, method: 'clearInterval' },
];

methods.forEach(({ object, method }) => {
    const isModified = isOverridden(object, method);
    console.log(`${method} is overridden:`, isModified);
});
```

Listing 6: Detection of overriden methods in various objects

Using this code, it can be determined which methods are most likely overridden. Before entering the Dynatrace script, none of these functions are reported as overridden. At the end of the script, the functions are reported:

```
addEventListener, removeEventListener, dispatchEvent, clearTimeout,
↪  clearInterval, open, send, fetch
```

Listing 7: Methods overridden by Dynatrace

When testing this on the previous two web pages with Hotjar and Lucky Orange, no methods are reported. However, this Dynatrace script is not certain to provide session recording, yet it is similar in some aspects. The company offers many analytics services apart from session replay, and it is not apparent which one this script uses. This could be one of the problems we might face, detecting session replay scripts without also flagging

other, less-intrusive analytics from the same provider. If all else fails, fallback methods can be used to detect the presence of known variable names used by session replay scripts or even the domains of these scripts.

In conclusion, common patterns can be extracted from the typical behavior of session replay scripts:

- initialization, configuration, session management,
- event handlers addition,
- data collection mainly using event handlers,
- data redaction and transmission.

From these stages of session recording script runtime, event handlers addition were the main focus of this exploration. To result in the reliable detection of session replay scripts, it must be accompanied by other indicators, which will be explored in the continuation of this project. The plan for this phase is to construct a heuristic for reliable detection. In the next steps, the script behavior should be studied on several session replay providers, ideally using all of the trial accounts mentioned in section 2.3.

Chapter 5

Conclusion

This project represents the beginning of a master's thesis with the goal of implementing the detection of session replay scripts in a browser extension. It summarizes current knowledge about session replay scripts, current possibilities of browser extension development, and the architecture of the extension JShelter. It proposes a methodology to describe and collect session replay script behavior patterns. This methodology is planned to be tested, revised and used to design the session replay script detection in JShelter.

Bibliography

[1] A complete guide - Glassbox. *What is Session Replay?* online. Available at: https://www.glassbox.com/session-replay/. [cit. 2025-01-12].

[2] Use Cases and Benefits [Guide] - Mouseflow. *What is Session Replay* online. Available at: https://mouseflow.com/topic-spotlights/session-replay/. [cit. 2025-01-12].

[3] Acar, G.; Englehardt, S. and Narayanan, A. No boundaries: data exfiltration by third parties embedded on web pages. In: *Proceedings on Privacy Enhancing Technologies*. Sciendo, 2020, vol. 2020, no. 4, p. 220–238. Available at: https://doi.org/10.2478/popets-2020-0070.

[4] Englehardt, S.; Acar, G. and Narayanan, A. *No boundaries: Exfiltration of personal data by session-replay scripts* online. Freedom to Tinker, 15. november 2017. Available at: https://freedom-to-tinker.com/2017/11/15/no-boundaries-exfiltration-of-personal-data-by-session-replay-scripts/. [cit. 2025-01-12].

[5] Filip, P. and Čegan, L. Comparing tools for web-session recording and replaying. In: IEEE. *2019 International Conference on Sustainable Information Engineering and Technology (SIET)*. 2019, p. 257–260. Available at: https://doi.org/10.1109/SIET48054.2019.8986134.

[6] Frisbie, M. *Building Browser Extensions: Create Modern Extensions for Chrome, Safari, Firefox, and Edge*. 1st ed. Berkeley: Apress, 2022. ISBN 978-1-4842-8725-5. Available at: https://doi.org/10.1007/978-1-4842-8725-5.

[7] Grodzinsky, F. S.; Miller, K. W. and Wolf, M. J. Session replay scripts: A privacy analysis. In: *The Information Society*. Routledge, 2022, vol. 38, no. 4, p. 257–268. Available at: https://doi.org/10.1080/01972243.2022.2078916.

[8] Hawes, C. Discover user pain points with session recordings. *What is session replay? Discover user pain points* online. Available at: https://www.dynatrace.com/news/blog/what-is-session-replay/. [cit. 2025-01-12].

[9] 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 (SECRYPT 2023)*. Rome: SciTePress, 2023, vol. 1, p. 287–294. ISBN 978-989-758-666-8. Available at: https://doi.org/10.5220/0011965600003555.

[10] Polčák, L. and Slezáková, A. Data Exfiltration by Hotjar Revisited. In: *Proceedings of the 19th International Conference on Web Information Systems*

and Technologies (WEBIST 2023). Rome: SciTePress, 2023, vol. 1, p. 347–354. ISBN 978-989-758-672-9. Available at: https://doi.org/10.5220/0012192500003584.

[11] Schwarz, M. L. M. and Gruss, D. JavaScript Zero: JavaScript and Zero Side-Channel Attacks. In: *Network and Distributed Systems Security Symposium*. San Diego, CA, USA: NDSS, 2018. Available at: https://doi.org/10.14722/ndss.2018.23094.

[12] Snyder, P.; Taylor, C.; and Kanich, C. Most Websites Don't Need to Vibrate: A Cost-Benefit Approach to Improving Browser Security. In: *2017 ACM SIGSAC Conference on Computer and Communications Security*. Dallas, Texas, USA: Association for Computing Machinery, 2017, p. 179–194. CCS '17. Available at: https://doi.org/10.1145/3133956.3133966.