TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technology
Department of Software Science

ITC70LT

Andres Rauschecker

USER-ORIENTED PRIVACY ENHANCEMENTS FOR WEB-BROWSERS

Master Thesis

Supervisor: Olaf Maennel PhD

Tallinn 2018

Author's declaration of originality

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication. All works and major viewpoints of the other authors, data from other sources of literature and elsewhere used for writing this paper have been referenced.

Author: Andres Rauschecker

May 6, 2018

Abstract

The numerous surveillance scandals and data leaks of our digital age show, that personal information has become one of the most valued goods online. International companies and governments seek to profit from exchanging their data records, leaving privacy rights in a critical state. As a great majority of digital content is accessed through web browsers, it is these applications, that require special attention in ensuring a user's privacy.
We show in a survey, that users often have difficulties in configuring browsers, so they match their security and privacy expectations. This is why we come up with a new defence concept, that addresses this issue: Based on analysing, how a user wants to use the Internet and what privacy aspects are considered important, we generate a custom configuration. Practically this is realised in a portable program.
Evaluating our new method, it becomes clear, that this way of setting up a browser improves protection, while hiding the complexity of technical details from the user. The enforced defence measures result in remediable rendering issues regarding video content. A more critical problem manifests itself in ensuring the trustworthiness of the application to the user. It is equally challenging to find a metric, that lets the user understand, how much protection he has in reality and how his answers influence this rating.

The thesis is in English and contains 62 pages of text, 5 chapters, 14 figures, 9 tables.

Annotatsioon

Meie digiajastul on toimunud arvukad jälgimise skandaalid ja andmete lekked, mis tõestavad, et isiklikust informatsioonist on saanud üks kõige väärtuslikemaid varasid. Rahvusvahelised ettevõtted ja valitsused püüavad kasu saada andmete vahetamisest, eirates seejuures indiviidide privaatsusõiguseid. Enamik digitaalsest infosisust saadakse veebibrauserite kaudu ning seetõttu vajavad just need rakendused erilist tähelepanu kasutaja privaatsuse tagamisel.
Uuringu tulemused on näidanud, et kasutajatel esineb tihti raskusi brauserite seadistamisel selliselt, et nende turvalisuse ja privaatsuse ootused oleksid täidetud. Sellepärast oleme töös esitanud uue kaitsekontseptsiooni, mis vastavat probleemi adresseerib. Analüüsi alusel, kuidas indiviid soovib kasutada Internetti ja milliseid privaatsuse aspekte peetakse oluliseks, on loodud mugandatud konfiguratsioon. See on realiseeritud portatiivse programmina.
Uue meetodi hindamisel selgub, et sellisel viisil brauseri seadistamine parandab kaitset, peites kasutaja eest keerukad tehnilised üksikasjad. Need kaitsemeetmed tekitavad video renderdamise probleemi, mis on siiski võimalik kõrvaldada. Suurem probleem on rakenduse usaldusväärsuse tagamine kasutaja jaoks. Samuti on keeruline leida mõõdik, mis võimaldab kasutajal mõista, kui palju kaitset ta tegelikkuses omab ning kuidas tema vastused mõjutavad hinnangut

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 62 leheküljel, 5 peatükki, 14 joonist, 9 tabelit.

Glossary of terms and abbreviations

API Application Programming Interface

CSS Cascading Style-Sheets

DNS Domain Name System

DOM Document Object Model

GUI Graphical User Interface

HSTS HTTP Strict Transport Security

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protcol

ISP Internet Service Provider

NSA National Security Agency

RFC Request For Comments

SOP Same Origin Policy

SQL Structured Query Language

TCP Transmission Control Protocol

TLS Transport Layer Security

XSS Cross Site Scripting

Contents

List of Figures

List of Tables

1. Introduction

1.1. Privacy in a current context

With Edward Snowden's revelations of global surveillance programs, that the American National Security Agency (NSA) used to monitor citizens world-wide [1], it becomes apparent, that personal information has become one of the most valuable resources online. It is not only governments, but also private businesses like Facebook or Google, who seek to sell the data they collect.
While actively user generated content, like submitted text or similar interactions, is the main source for data collection, meta-information also plays an important role in analysing a user's behavioural patters: Knowing where a visitor came from and what pages he has viewed in the past, allows to put a current action into context. It becomes possible to link identities between services and understand how single requests correlate to each other.

1.2. Research problem

As a majority of today's Internet traffic is exchanged via web browsers, these applications become a central target of the previously described meta data analysis. Browsers have developed into complex programs and many components, designed to enhance functionality, can be exploited to gain additional information about their user. Looking at the past decade of security incidents, we can see numerous issues with applications like Firefox or Chrome and third party plug-ins like Adobe Flash [2]. Secondly, the amount of techniques used to track and control users across the Internet has grown significantly. Companies want to know, what advertisements potential customers clicked on. Governments have recognized, that the digital world offers a new space for illegal activities and thus want to monitor online behaviour.
While institutions like the TOR Project [3] and several private contributors develop tools to protect against the latest attack trends, in the end it is the user, who has to decide, which software is working as intended. He has to install a set of applications or plug-ins, that fit to his privacy and security needs. As previously mentioned, this task becomes more and more complex, as one has to understand how attacks work and analyse, whether a tool correctly defends the latter.

1.3. Research question

This thesis evaluates, how browsers can be made more secure and protective to privacy, by including the user in the configuration process: Can a better understanding of a user's intents improve the way how browsers are set up? Subsequently, this research question is structured into the following aspects:

1. Defining privacy and security. What do we aim to protect against?:
   We discuss, where the identity of a user is reflected in his browser and in what known ways it can be revealed or tracked.

2. Who is the attacker? How may he compromise our goals?
   As mentioned previously, several entities aim to benefit from collecting and linking meta information about their users. We will analyse their motives and options.

3. How are devices configured in reality?
   Via an online survey, we will dissect browser configurations and evaluate, how they match their users' privacy intents.

4. How can we reduce configuration complexity and meet the user's needs?
   We attempt to extract individual preferences from the user and use this information as a foundation to allow an automated configuration of web browsers.

5. How much can we improve protection? What limitations exist?
   We quantify our adaptations and evaluate, how effective usage-based defence can be. We also look at possible restrictions (e.g. usability and granularity of the defence).

1.4. Limitations

Because of the limited time frame for the thesis, only Mozilla Firefox will be used and analysed within this work. The research outcomes are however still applicable to other browsers, as most of them share the same attack surfaces and offer similar configuration options for a defence. For the application implemented in the scope of the thesis, Google Chrome and Microsoft Edge support are planned.

1.5. Outline

The thesis is structured as follows: Chapter 2 will find a definition of digital privacy, discuss attacks to it and select methods for a successful defence. Then, chapter 3 will analyse in a survey, how users configure their browsers and what privacy aspects they consider important. Subsequently, chapter 4 focuses on the realisation of a configurator application, that intends to enhance individual privacy by questioning the user about his browsing intents. In the last sections, it will evaluate the benefits and limitations of the created software. Lastly, chapter 5 will summarize the research and conclude on the pros and cons of the user oriented application, as well as the idea behind it. This includes a future work section, where extensions to the work are discussed.

2. Related work

Before we can develop a methodology on how to include user intents in browser configurations, it is necessary to give a clear understanding of the terms, which will be used in the thesis. Secondly, we will look at possible attackers, their motivations and how they can interfere with our defence goals.

2.1. Definition of privacy and security

As Onn et al. recognize in their publication "Privacy in the Digital Environment", privacy is a complex term, that is dependent on many parameters, such as time and context [4, p. 6]. This is why it seems less practical to cite rigid legislation, but rather use a more open definition and translate it to the digital space, we will be focusing on. While the previously mentioned work aims to discuss the current state of online privacy and possible enhancements to it, the authors come up with a very considered definition [4, p. 12]:

> The right to privacy is our right to keep a domain around us, which includes all those things that are part of us, such as our body, home, property, thoughts, feelings, secrets and identity. The right to privacy gives us the ability to choose which parts in this domain can be accessed by others, and to control the extent, manner and timing of the use of those parts we choose to disclose.

To adapt the personal domain of the first sentence to a digital world, it is useful, to think through example interactions with a website: A user can request a page, enter his own thoughts into input fields and send them back as an answer. Looking at the personal information he shares, it is his thoughts, that he decides to disclose, as well as his identity, that is linked to his IP address or authenticated session parameters.
Understanding the second sentence in a digital context, a common page request can be considered a privacy agreement, where the user grants the server access to his identity and the requested page name for the purpose of obtaining the requested content. A user-generated input submission constitutes a similar privacy agreement: The user allows the server to obtain his thoughts and identity within the scope, that the input page displayed.

Summarizing these elements, we can update the original privacy definition, to form a foundation for our further research: Digital privacy is the right to choose which personal information can be accessed by others. Every user-server interaction denotes an agreement, where a user grants access to his identity and/or thoughts. The agreement is bound temporally for the duration of the enclosing interaction and is access limited to the parties, that are part of the interaction itself.

The term security will be used in close relation to privacy: Whenever a privacy agreement is violated by breaching security concepts, this attack surface will be considered necessary to defend.

2.2. Attacker model

As a next step it is necessary to analyse, who wants to interfere with a user's privacy or possibly his web-browser's security, to benefit from. To do so, we will extract attacker types from recent literature and discuss their motivation. Secondly, we will take a look at the attack surface of web-browsers and go through currently known attack techniques. Finally we will match these attacks to attackers and evaluate defence strategies against each of them.

2.2.1. Malicious hackers

As a first attacker type, we can identify individuals or smaller groups, that mostly target the aforementioned security concepts of web-browsers to gain profits. The motivation is partially monetary: Hackers may sell unknown exploits in the Darknet or exchange them against other goods, that are of interest. [5, p. 12, section 2.1.1.2] They may also be paid by thirds to attack someone, like it is often the case in industrial espionage or political conflicts. Another incentive is to be found in propaganda and digital ideologies. An example is Anonymous: As Fuchs states in his publication "The Anonymous movement in the context of liberalism and socialism" [6, p. 347], the collective makes use of a shared identification with values like freedom of media, as well as fighting for justice and freedom. Some hackers thus act solely out of the belief in certain values.

2.2.2. Governments

Like already mentioned in the introduction, Snowden's revelations of the NSA's surveillance programmes, have made it clear, that governments spy on their citizens. While these acts are commonly declared to be part of anti-terrorism strategies or homeland defence, we can see, that in some countries, citizens are actively attacked, in order to hold up governmental power: Al-Saqaf shows in his dissertation, that during the Arab Spring, several regimes used malware to maintain Internet censorship [7, p. 206]. A third motivation for the control of the online space can be found in law enforcement: Actions like breaking copyright law or uploads of prohibited content require a government to invade privacy rights, so perpetrators can be found and made responsible for their legal violations.
Looking at the resources of governments, we can see a high amount of capital, personnel and time, as well as access to the military sector and cooperation with Internet Service Providers (ISPs).

2.2.3. Advertising companies

Online advertising allows publishers to benefit from obtaining several details about their users. Advertisers know what exact contents are viewed and can target their messages very specifically. [8, p. 42] A core technique to get such a deep understanding of website viewers lies in third-party advertising: Website providers embed advertisement elements from other companies and share information about the current page and viewer with this element. This allows the advertisement company to track, what sites and thus possible interests a user has, which improves ad targeting. [9, p. 415] This goes against our definition of privacy rights, since identity and personal information are communicated to another party, that the user has not directly agreed to exchange data with.
The motivation for this business model is clear: First-party websites can make money by embedding advertisements. The tracking companies generate revenue by selling advertisement spots to businesses, who want to promote their products. Ultimately, these businesses increase their profits, since the targeted ads encourage end customers to buy their goods.

Resource wise we can see, that the tracking data is sold amongst each other [9, p. 419], which extends the data sources that advertisers work with. However, they can not acquire information from ISPs, to link identities and IP addresses.

2.2.4. Conclusion and scope of defence

We listed governments, advertising companies and malicious hackers as the three main groups of attackers, that are trying to invade our privacy. As we want to develop a defence scheme in the following sections, we need to assess, what adversaries we can realistically protect against [10, slide 15-16]: It is apparent, that governments have a big capital and connections to ISPs and the military. This means, that they can buy zero-day exploits and exchange with security agencies world-wide. They have access to ISP logs and Internet backbone infrastructure. Protecting against this adversary is highly complex. Even if we manage to reduce our browser fingerprint to a minimum and deploy various protection mechanisms, they can use one of the many mentioned information channels to circumvent our tactics. Thus a defence attempt seems impractical in the scope of a thesis. Looking at advertising companies and (sponsored) hackers, their motivation is mainly monetary: They do not care, if single browsers can not be tracked, as long as millions of other users generate revenue by loading their advertisements and feeding into their tracking systems. Thus this opponent seems to be better suited for our research: By reducing accuracy of fingerprinting and tracking, we can prevent personal data to be sold and provide users a better protection for their data. Out of these reasons, the following sections focus on a defence against these commercial parties, that prioritize mass-data harvesting.

2.3. Known attacks

This part will discuss currently known attacks, that interfere with or circumvent the privacy model, that we previously defined. Additionally, we will talk about ways of measuring the risk, that a browser is exposed to, in order to allow a comparison of protection before and after developing a defence.

2.3.1. Exploits

As mentioned in section 2.2.1, the term "exploit" refers to a manipulation of a program state, that leads to a breach of security mechanisms in place. Since browser applications are extensible and use features from many different sources, attacks can be executed on the following elements:

2.3.1.1. Browser extensions Browsers like Firefox or Chrome allow the user to install extensions, which are inserts, that can communicate with the browser functionality over a defined interface. They can provide different functionalities like ad-blocking or bookmark storage. As Liverani and Freeman show in their talk on "Abusing Firefox Extensions" [11], these extensions are sometimes not checked as thoroughly or their complexity makes it difficult to find all possible vulnerabilities, which leads to attack opportunities on the extensions themselves. While Mozilla has made many changes to this infrastructure, like disallowing unsigned software [12, section Timeline], there are still risks of undiscovered flaws. Additionally, an article on Chrome browser security shows, that the developers themselves can become victims of phishing attacks, which lead to a compromise of several Chrome extensions in 2017. The attackers subsequently injected their own, malicious Javascript into the applications, which the browsers then executed. [13]

2.3.1.2. Plug-ins The set of problems, that extensions come with, similarly exists for browser plug-ins: Software technologies like Flash or Java have been known for the many security issues, in connection to their browser plug-ins. Their exploitation very often leads to a full browser or device compromise, like CVE-2012-0507 shows: A flaw in the Java plug-in allowed malicious hackers to install botnet malware on the system [14].

2.3.1.3. The browser itself Looking at vulnerability databases like CVE Details, we can see that Chrome and Firefox are amongst the top ranks for the number of distinct vulnerabilities found [2]. With complex features like the Same Origin Policy (SOP) restriction or protections against Cross Site Scripting (XSS), edge cases in parsing and other bugs can easily lead to a security flaw, that compromises the browser's security.

2.3.1.4. Measurement As Acer and Jackson describe in their publication on browser security metrics [15, p. 3], it is no good practice to measure exploit risk by just counting the number of reported vulnerabilities per extension, plug-in or browser version, since this does not take patch development or the severity of the flaws into account. They propose to use an adapted risk score: Products or versions with at least one critical vulnerability are relevant, others are not. (This uses the CVSS v.3.0 Severity Rating System, as explained in [16].) We want to adapt this filtering and will attempt to block or protect browser products, that qualify according to these metrics.

2.3.2. Fingerprinting

As Laperdrix, Rudametkin and Baudry explain, browser fingerprinting is the process of collecting browser or system configuration data, that is then used to extract unique values, which allow the construction of a unique identifier. [17, section I] This has severe implications for the privacy of a user: Instead of being identifiable by the IP address or deliberately shared authentication tokens, webpages can use fingerprints to follow identities and browsing behaviour. Since the success of fingerprinting techniques depends on the amount of entropy, that can be read from configuration parameters, it is necessary to evaluate individual methods and their uniqueness, in order to understand the defence process. The following paragraphs use data from Buljow et al. "Web Tracking: Mechanisms, Implications, and Defenses" [18, p. 4, table II, section V] and will explain the most relevant fingerprinting techniques more closely:

2.3.2.1. Device If the user's browser has Adobe Flash installed and is permitting the use, a website may obtain information about hardware, including mouse, keyboard, accelerometer, touch functionality, microphone, camera or screen resolution data. The Flash platform itself provides these access methods. [18, p. 9, section B]
A second device fingerprinting method lies in the analysis of Transmission Control Protocol (TCP) timestamp clock skew: A machine has a specific time offset, as compared to a master time server. This offset stays constant and can be evaluated. However, it is not precise enough to reveal identities or track users, since multiple devices can have the same offset, especially when they were synchronized to a network time server. [18, p. 9, section B]

2.3.2.2. Operating system Looking at operating system parameters, that create entropy, we can find several data points via Javascript: We can identify the operating system version and architecture, as well as system language, user-specific language, local timezone, date, screen resolution and screen color depth. [18, p. 9, section C]
Available system fonts can be found by using Flash plug-ins. An alternative method is the analysis of fonts loaded via a CSS3 font-face rule in combination with Javascript. A third technique is based on rendering fonts inside HTML5 canvases and evaluating the results. [18, pp. 10-11, section E]
Next to the mentioned multimedia capabilities of the previous paragraph, Flash also allows to check, if hard disk or printing access has been granted. [18, p. 9, section C]

Another plug-in, that was found in our source analysis, is Active X, a proprietary interface from Microsoft. Since we focus on the open Mozilla Firefox in this thesis, we will not further investigate this technology. (It is by default not supported.)

2.3.2.3. Browser Further entropy can be extracted from HTTP responses: One can enumerate, what file types are supported by the browser and what preferred or accepted languages are set. The User Agent HTTP header field also reveals the browser's version. Using Javascript, there is additional ways, how identification can be enhanced: Firstly, the browser version can be determined by analysing CSS and HTML5 properties, that are specific to certain releases. Secondly, a list of installed plug-ins can be obtained. [18, p. 10, section D] Another creative technique to determine, if a website has been visited, lies in evaluating the cache state of website elements: Pages, that have already been visited, are usually existent in the browser cache, which reduces loading time measurably. [18, p. 7, section IV] Lastly, a combination of HTML5 canvas elements and Javascript allow the generation of very high entropy: Certain shapes and text fields are drawn onto the canvas. The browser renders this image uniquely, as it is dependent on the used graphics card, installed fonts, operating system, drivers and other components. The same technique can be realised with the WebGL Application Programming Interface (API) [18, pp. 10-11, section D]

2.3.2.4. Measurement Regarding the measurement of fingerprinting techniques one possibility would be the analysis of anonymity set sizes: This means finding out, how many browsers have the same fingerprint. One downside however is, that this metric depends on the amount of samples collected, as it is impossible to predict, how fingerprinting parameters are distributed on the Web. Laperdrix, Rudametkin and Baudry propose a better means of quantifying fingerprints, namely normalized Shannon's entropy [17, pp. 5-6. section C]: Shannon's entropy is defined as

$$H(X) := -\sum_i \mathbf{P}(x_i) \log_b \mathbf{P}(x_i)$$

where $b = 2$ and thus the result in bits. This unit is still based on the overall amount of samples. To get a value, that is comparable with different sizes, the entropy is divided by the maximum entropy of the dataset size (N):

$$\frac{H(X)}{\log_2(N)}$$

This normalized entropy is used by the previously mentioned authors and will be adopted in our defence discussion as well.

2.3.3. Tracking

Tracking describes the process of monitoring the browsing behaviour of a user: The goal is to find out, where a visitor comes from, where he goes next and what actions he possibly performs on a page. While fingerprinting has a more passive nature, trying to find unique parameters of the browser and its environment, tracking often stores identifiers on the target device or actively communicates tokens between pages. The most common forms of tracking will be explained as follows:

2.3.3.1. Session identifiers The most basic form of tracking lies in passing an identifier via GET or POST request to the next page. While GET parameters are saved within the history of the browser, the POST context will disappear after the transmission of the request packet. A similar variant of this storage-free tracking uses the window.name Document Object Model (DOM) element, which can store up to 2 MB of page values and is resistant to reloading. In most cases, every visit of a page will generate a new session identifier and the page owners have no means of linking previous page access to the current one. [18, pp. 3-4, section A]

2.3.3.2. Cookies A more advanced way to track a user utilizes cookies. Cookies are objects, that reside inside a browser's internal storage. They can be set on a per-page basis and will continue to be stored, even after a restart of the computer, provided, that the user does not manually clear the cookie memory. Cookies can be combined with the previous methods, to associate an authenticated identity with a tracking ID (via post forms) or to communicate the ID to another website. The latter can not be done by the third-party website, as the domain limitations of the cookie standard forbid access to values, that do not lie within the own domain's scope [19, section 4.1.2.3].

While cookies have been defined in the Request For Comments (RFC) 6265 document, plug-in developers have created independent storage concepts for their own products. This is why Adobe Flash allows pages to store data in so called Local Shared Objects. It is interesting to note, that these items are accessible from all browsers on a machine and

thus have a system wide scope. They also have no default expiration date and are written to disk memory. Similarly, Microsoft's Silverlight plug-in provides isolated storage on a per site basis. This store can only be deleted manually. The newer HTML5 also comes with Local Storage objects, which does not require any plug-in to be installed. They are cleared when emptying the standard browser cookie store. [18, pp. 6, section B]

2.3.3.3. Caches Another technique, that can be used for tracking, looks at data inside browser caches. The intended purpose for these memory units is for pages to load faster: Some page sources are saved and kept in memory. On a new request, these items do not need to be downloaded again, since they are still available. In a tracking scenario, these local files are used to analyse, if a page has been visited before and to extract user identification data. For the former, we can use Javascript to profile the loading time of a page. Since already cached pages or page elements speed up the loading process significantly, this speed improvement can be measured by the server. A similar variant of this technique uses Javascript to make Domain Name System (DNS) requests. Already cached DNS entries result in faster responses. It should be noted, that this technique uses the operating system's DNS cache, instead of a browser cache. Determining if a page has been loaded before can also be achieved by analysing HTTP 301 redirect caches: In the case, that a page has moved permanently to a new place, the browser will receive a permanent redirect notice by the server. Every new request will be made directly to the new address. If a browser thus requests only the new page location, we know, that it must have visited the website before. [18, p. 7-8]
In order to obtain a user identifier, the following methods exist: Firstly, an identifier can be embedded into an HTML file. After the page has been loaded from cache, this token can be accessed via Javascript. Instead of hiding the identifier in the code itself, it is also possible to transfer them within the Last-Modified and ETag fields of an HTTP header. These parameters are usually used to identify, if cached content is outdated and needs to be reloaded. Because of their size, identifying tokens can be embedded, though. The most sophisticated technique to store identifying values is using the HTTP Strict Transport Security (HSTS) cache to do so: The cache is originally used to store, that all future connections to a host shall be made using HTTPS only. A server can communicate this information with a special header parameter. The parameter allows to specify, what subdomains are included for the security policy. An attacker can take advantage of this by creating fake subdomain values, that encode a user ID. This ID can later be brute-forced via Javascript, by querying, what subdomain values are directly visited in HTTPS and which ones still use HTTP. A more detailed explanation can be found in [20]. A last tech-

nique, that has been described in [18, p. 7-8] is Transport Layer Security (TLS) Session cache analysis: A browser keeps another cache with HTTPS session tokens, which allows him to establish future connections without having to execute a full TLS handshake, which ultimately saves CPU and loading time. A page can monitor, whether these IDs are re-used and thus infer, if a prior connection has been made or not.

2.3.3.4. Measurement It is impossible to find out, if an identification token on a webpage is used to authenticate a user for a legitimate page interaction or if that token is exchanged with external parties to allow tracking. This means, that there is no clear way of counting every single tracking element on a page and evaluating, how well defence concepts work. However, we can focus on tracking, that is distinct in its intents and can be identified as such: Roesner, Kohno and Wetherall observe in their paper "Detecting and Defending Against Third-Party Tracking on the Web" [21, p. 8, table 3], that the majority of pages they analysed (Alexa Top 500 from Sept. 2011) is using third-party storage based tracking mechanisms. As previously described, this technique stores a cookie or a similar object on the hard drive, which will commonly stay available after a browser restart. We want to measure this persistent type of tracking and its defence by looking at the browser storage memories and comparing the results with filtering tools in use. We still want to discuss, how session identifier tracking and cache based tracking can be mitigated, but since most of these techniques are not detectable from outside of a server, it is not possible to visualize defence efficiency.

2.4. Defence strategies

In the following we will evaluate, how the previously described attack types can be defended. This chapter will be the foundation for our own browser configuration application: Ideally we can find a set of tools and settings, that protects against each technique. We can then combine this set with a user's privacy preferences to generate an adapted browser configuration.

2.4.1. Risk reduction for exploits

Based on the exploit risk level we defined in section 2.3.1.4, we now want to analyse, what browser elements we need to block, in order to minimize the probability of our browser getting compromised.
In the category of browser extensions, we want to refrain from using proprietary software. Additionally, it seems a good practice to integrate only actively maintained projects: While several smaller contributors to Mozilla's extension library exist, these do not have the capacities to quickly respond to security issues. Often the projects are outdated and support is not guaranteed.

Looking at the security of plug-ins, Mozilla has already taken measures against a big part of insecure software, by dropping NPAPI plug-in support in Firefox version 52 [22]. This means plug-ins like QuickTime, Java or Adobe Reader can not be used any more. However, Mozilla still allows the use of Adobe Flash and Microsoft Silverlight, as the only supported plug-ins. In section 2.3.2.1 and 2.3.3.2, we have found, that both these plug-ins allow or extend fingerprinting and tracking of the user, while additionally being known for their security risks [2]. This is why we want to disable these plug-ins, if the user does not explicitly signal the need to use them.

Firefox allows to set the plug-in activation state via its internal configuration system [23]:

```
plugin.state.<name> → 0 | 1 | 2
0: Never Activate, 1: Ask to Activate, 2: Always Activate
```

We can thus disable the Flash and Silverlight plug-in, by setting the following values:

```
plugin.state.flash → 0
plugin.state.silverlight → 0
plugin.state.npctrl → 0 // Windows specific (Silverlight)
```

If we store these values in a separate Javascript file and place them inside the install subdirectory (as specified in [24]), Firefox will execute our settings upon application start.

Regarding the security of the Firefox browser itself, we can make several adjustments via the same configuration interface. The GitHub repository at [25] has been compiling an extensive collection of parameters and optimised values since 2014, while still being

maintained as of March 2018. We will extract settings, that are relevant for our exploit risk reduction in the following:

- As [25] mentions, DOM workers have had multiple vulnerabilities in the past. We thus want to disable this technology.
- The WebTelephony API might be used to call high cost numbers or record phone calls, which we want to prevent [26].
- As [27] shows, numerous security issues have been found in webGL, we therefore disable this feature set.
- Unsafe JAR file types are blocked, in order to prevent a XSS attack possibility [28].
- The asm.js Javascript subset has had some critical security issues, as to be found in [29]. Thus we will disable it.
- Shumway is Mozilla's own Flash renderer. We turn off Flash, like previously explained and want to block this implementation as well.
- *plugins.click_to_play* allows us to force the user to explicitly enable a plug-in. This option is improving security, as plug-ins can not execute without permission.
- We enable automatic extension updates.
- The firefox configurator allows to enable unsafe extension blocklists. We enable this feature to protect the user from installing untrusted or insecure extensions.
- Disable the built-in PDF viewer, as some vulnerabilities have been found (see [30]).
- Enable automatic updating and update checks.
- Block reported webpages with malicious content.
- Enable Firefox Content Security Policy. This defends against some data injection and XSS attacks.
- Enable Subresource Integrity. This makes sure, files fetched from Content Delivery Networks can not be forged.
- Disable insecure versions of TLS.

Summarizing these items, we get the following list of key pairs, that need to be set:

```
user_pref("dom.serviceWorkers.enabled", false);
user_pref("dom.workers.enabled", false);
user_pref("dom.telephony.enabled", false);

user_pref("webgl.disabled", true);
user_pref("network.jar.open-unsafe-types", false);
user_pref("javascript.options.asmjs", false);
user_pref("shumway.disabled", true);
user_pref("plugins.click_to_play", true);

user_pref("extensions.update.enabled", true);
user_pref("extensions.blocklist.enabled", true);
user_pref("services.blocklist.update_enabled", true);
user_pref("pdfjs.disabled", true);
user_pref("app.update.auto", true);
user_pref("app.update.enabled", true);

user_pref("browser.safebrowsing.enabled", true);
user_pref("browser.safebrowsing.phishing.enabled", true);
user_pref("browser.safebrowsing.malware.enabled", true);

user_pref("security.csp.enable", true);
user_pref("security.sri.enable", true);

user_pref("security.tls.version.min", 1);
user_pref("security.tls.version.max", 4);
user_pref("security.tls.version.fallback-limit", 3);
```

2.4.2. Anti-Fingerprinting

In section 2.3.2 we have compiled the most common fingerprinting techniques, that are in use. In order to defend against them, it is useful to look at the amount of entropy each of them generates: Analysing, for instance, if cookies are enabled, yields two distinct values: either yes or no. Revealing this one bit of entropy to an attacker will not make us unique within the anonymity sets, that the Internet consists of, thus it is irrelevant to block this parameter.

Laperdrix, Rudametkin and Baudry have made an evaluation of normalized Shannon's entropy per attribute (see 2.3.2.4), that we want to order and adapt for our purposes:

Attribute	Norm. Ent.	Example
List of plug-ins	0.718	Plugin 0: WebKit built-in PDF; ; .
User agent	0.550	Mozilla/5.0 (Macintosh; Intel Mac OS X [...]
List of fonts (Flash)	0.548	Abyssinica SIL,Aharoni CLM
HTML5 Canvas	0.475	190E04503900C8FB00612E06D6929400
Content language	0.344	en-us
Screen resolution (JS)	0.263	1920x1080x24
List of HTTP headers	0.247	Accept Cookie Accept-Language [...]
Renderer WebGL	0.205	Intel(R) Iris(TM) Graphics 650
Timezone	0.200	-60 (UTC+1)
Vendor WebGL	0.125	Intel Inc.
JS Platform	0.110	MacIntel

Table 1. Normalized Shannon's entropy of Fingerprinting methods
(adapted from [17, p. 14, appendix A] and [17, p. 4, table I])

As we can see, the list of plug-ins has the highest amount of unique values. Fortunately, Mozilla's drop of NPAPI plug-ins from version 52 [22] reduces the entropy of this attribute. We can furthermore minimize distinct values by setting the Firefox configuration parameter *plugins.click_to_play* to **true**, since this removes all plug-ins (except Adobe Flash) from the Javascript enumeration interface [31]. Going on, we have to look at ways of making the user agent less unique. Paradoxically, faking the most prevalent value can make a browser more unique, since the behaviour of its components does not match the fake value: We can not simulate a Windows system, when we are using macOS, since this is easily detectable. Further investigating Firefox's configuration possibilities, we are able to find the following: From version 55, Firefox contains the key *privacy.resist.fingerprinting*, that, once set to **true**, will activate defence against a variety of fingerprinting techniques [32]. These include:

- User agent: Changing the browser version to the last Extended Support Release.
- Fonts: Limiting system fonts to a restricted subset.
- HTML5 Canvas: Blocking Canvas image extraction entirely.
- Content language: Advising the user and setting *en-us* as the default.
- Screen resolution: Setting the screen resolution to the window dimensions, rounding window dimensions to a multiple of 200x100. → This creates unrealistic screen

resolutions, thus we want to use the parameters *privacy.window.maxInnerWidth* and *privacy.window.maxInnerHeight*, to simulate a more common 800x600 screen. This is the best option, since individual defence elements can not be turned off.

- Timezone: Is changed to UTC time.

Comparing this with table 1, we can see, that only the list of HTTP headers, WebGL and the Javascript platform are excluded. We disabled WebGL in section 2.4.1, which leaves us only with HTTP headers and the Javascript platform. For these last two attributes we could not find a suitable defence: Limiting HTTP headers would require to block core functionality within Firefox, since some HTTP headers are being used for advanced communication with a server (e.g. X-DNS-Prefetch-Control). We could not find browser extensions or configuration settings, that modify this functionality. Spoofing the Javascript platform version is also not possible, since many functions are directly platform dependent. We would have to limit the entire function set, in order to plausibly fake this value.

2.4.3. Anti-Tracking

Going through section 2.3.3, we can see, that tracking uses three different approaches: Firstly passing identifiers via GET or POST, secondly, using various forms of cookies to store this data and thirdly using browser caches as a storage alternative. Filtering GET or POST requests for tracking data is not very effective, as it is hard to determine, if a parameter is a unique tracking ID or a regular value. Because of this, we will look at a different defence approach: Tracking services rely on the concept, that their tracking script is embedded by a large number of servers. As long as subsequent pages embed the same script, the tracking company is able to track the history of a user. This tracking system is enhanced by several trackers sharing their databases amongst each other. It is highly impractical to frequently create new tracking systems, because pages need to embed the new scripts, in order for this tracking to work. This means, that a continuously updated blacklist is able to block most tracking services. Again, exploring the configuration options of Firefox itself, we can find a tracking protection mode, that uses a blacklist to block known tracking sites [33]. By modifying the *urlclassifier.trackingTable* value as follows, the *strict* blocking list is used, which blocks all trackers known to Firefox:

```
privacy.trackingprotection.enabled → true
privacy.trackingprotection.pbmode.enabled → true
```

```
urlclassifier.trackingTable →
   test-track-simple,base-track-digest256,content-track-digest256
```

This is a good alternative, also to compensate for the missing GET and POST filtering. Next, we want to take a look at cookie tracking and how to protect against this method:

A basic measure is to only allow cookies from first parties. This means cookies, that are not matching the domain name of the active window are blocked. Trackers can circumvent this by opening third parties in a popup or redirecting to them [18, pp. 5-6]. Thus we want to take further adaptations to limit tracking mechanisms: We want to set cookies to be valid in the current session only [34]. This means they will be cleared, if the browser is closed. Furthermore we block Flash cookies by completely disabling Flash. We also block HTML5 and DOM storage by setting another configuration value [35]. Going through the configuration possibilities of Firefox, there is an additional, useful item to make tracking harder: First party isolation. This feature was adopted from the TOR browser and not only limits cookies to the first party domain, but also caches, HTTP authentication and DOM storage [3, section 4.5]. This covers all of the points we discussed in section 2.3.3.2. The summarized Firefox configuration for enhanced cookie tracking protection thus looks as follows:

```
network.cookie.cookieBehavior → 1 // first party only
network.cookie.lifetimePolicy → 2 // session only
dom.storage.enabled → false // disable DOM store
privacy.firstparty.isolate → true // first party isolation
```

As a last tracking protection part we want to cover cache based methods. Following [25], we firstly want to disable offline page caching, by setting *browser.cache.offline.enable* → **false**. Furthermore we want to remove all histories, caches and local storage upon browser shutdown. This will also clear the redirect cache as well as TLS session IDs.

```
privacy.sanitize.timeSpan → 0 // clear the entire history
privacy.sanitize.sanitizeOnShutdown → true
privacy.clearOnShutdown.cache → true
privacy.clearOnShutdown.cookies → true
privacy.clearOnShutdown.downloads → true
privacy.clearOnShutdown.formdata → true
privacy.clearOnShutdown.history → true
privacy.clearOnShutdown.offlineApps → true
```

```
privacy.clearOnShutdown.sessions → true
privacy.clearOnShutdown.openWindows → true
privacy.cpd.offlineApps → true
privacy.cpd.cache → true
privacy.cpd.cookies → true
privacy.cpd.downloads → true
privacy.cpd.formdata → true
privacy.cpd.history → true
privacy.cpd.sessions → true
browser.helper.Apps.deleteTempFileOnExit → true
```

We can completely disable site caching by deactivating the caches themselves:

```
browser.cache.disk.enable → false
browser.cache.memory.enable → false
```

In order to prevent DNS cache tracking, we can disable this cache entirely [36]:

```
network.dnsCacheEntries → 0
network.dnsCacheExpiration → 0
```

We also want to disable auto filling HTML forms: *signon.autofillForms* → **false**.

The last tracking technique, that we wanted to address is HSTS cache based tracking. As HSTS itself is used to defend against protocol downgrades and more severe attacks (e.g. Man-in-the-Middle traffic interception), we do not want to disable or clear this cache and need to accept the tracking risk.

2.5. Current state of defence strategies and our contribution

Looking at existent research in the field of webbrowser privacy, we can see, that sources like [17] or [18] have great depth, describing various attack concepts. As [18, p. 17-24] clarifies, there are several tools available, that defend against one particular attack method, however, there is only very few, that combine defence for multiple attack vectors. The latter exclusively share a commercial background, which means, the user has to pay, in order to use a proprietary (thus closed-source, non-transparent) solution.
Our approach has two main contributions to the existing set of defence projects: Firstly, we implement an open-source application, that combines free-to-use tools and browser configurations. Our tool therefore is a first alternative to using the paid applications. Secondly, we attempt to include the user in the configuration process. The majority of tools (e.g. AdBlock Plus, uBlock Origin) provide only an "on/off" interface, which leaves actual functioning and configured defence unknown to the user. During our literature and research analysis, we also found, that user-integration in the field of web browser privacy has not been focused on particularly, which is why our work contributes to a better understanding of this matter, by answering questions like: Can our idea allow for a finer-grained adaptation of individual privacy demands? What limitations and challenges does this way of configuration result in?

3. Survey: Analysing current browser configurations

In the previous chapter we have dealt with the definition of digital privacy. We compiled a list of attackers and various attack types. Lastly we investigated, how a browser can be configured in order to mitigate or protect against these risks. After gaining an understanding of these various elements we want to proceed with our research question:

Can we improve browser privacy by integrating the user in the configuration process?

In order to do so, we want to analyse in a survey, how current browser configurations look like and also how participants view online privacy, as well as our approach of implementing a configurator application.

3.1. Survey design

We want to split the survey into two parts: One dedicated to information about the user, the other focusing on the technical details of the corresponding browser.

3.1.1. User data

Firstly, we want to know the current work position and institution of the participant. We can use the work position to understand, what the tasks of the user are.

As a next point we want to know, what media content needs to be viewed while browsing. Since we disable Flash and other plug-ins, it is necessary to evaluate, if these technologies are still in use by some individuals and we thus need to find alternative configuration options. Some of our settings (like disabling DOM workers) also limit the functionality on pages like Google Docs, which we possibly need to adapt to. The different media types we question are: *Video, Flash, Java, Browser Apps, Interactive Chats*

After analysing the tasks and contents, the participant interacts with, we want to find out, what privacy aspects are considered important. We made a list with the following options (and explanations to them):

Fingerprinting, Anti-Tracking, Adblock, IP Anonymity, Highest possible risk reduction, Anti-Cryptocurrency Mining

This covers our three main defence points, as well as some additional functionalities we can think of.

Another essential question we wanted to ask is: How likely would you use an external program to configure your browser? Here we try to understand, how the trust relationship to an application, like the one we want to develop, is. The participant can answer on a scale from one to five: very likely to not at all likely. Additionally, we offer to provide an e-mail address, to participate in beta testing, once our program is ready.

3.1.2. Browser data

Regarding the browser settings we want to find out, if any parameters have been customized and if so, what changes have been made. Firefox can be configured using configuration files or additional tools like plug-ins and extensions.

The main file, configuration changes are saved in, has the name *prefs.js* [24]. Out of security reasons the file content is not accessible online, which is why we ask the user to manually upload it.

It is equally impossible to enumerate extensions from a browser. Because of that, we also ask the user to submit a list of all installed extensions, that can be found in the *about:support* section of Firefox.

3.2. Outcomes

Since the limited time frame of our thesis did not allow us to gather enough votes for a statistically significant evaluation, we will focus on some qualitative observations in our result set. A more detailed version of the survey results is attached in Appendix A.

3.2.1. Usability versus protection

Some of the answers showed, that technologies like Adobe Flash were still used. As we saw in table 1, enabled Flash greatly increases the entropy for Fingerprints. Equally, blocking DOM workers, like we do in our exploit reduction section, limits some of the functionality of pages like Google Docs. These online apps were frequently marked to be used by the participants.
This means we have to create an option to enable these technologies, while clearly explaining to the user, how the choices impact his browsing experience and privacy protection.

3.2.2. Trustworthiness of an external application

Secondly, we received mixed answers regarding the desire to use an external application to configure one's web browser. Even if we publish the source code of our program, it will still be hard for non-technical users to verify, that it works and that is does not do anything else.
For the testing phase we can omit trust problems, but in case the software becomes advanced enough to work with several browsers and usage scenarios, some kind of external verification is beneficial.

3.2.3. Privacy intents versus actual configuration

Most interesting is the relation between the desired privacy aspects and how they are configured in the browser: The participants who signalled, that they would like to be protected against tracking, fingerprinting or advertisements often had very different configurations: The typical pattern we found, is, that one main browser extension (like uBlock Origin, McAfee Endpoint Security Web Control or Browsing Protection by F-Secure) enforces basic protection against malware and unsafe sites via blacklist approach. The browser configurations themselves did not contain any relevant changes.
This shows, that in our limited answer set, one of our research assumptions is true: The configuration of web browsers is complex and often the users believe, that one tool will be sufficient. **The actual protection and initial privacy intents do not match**. This leaves a false sense of security.

4. Implementation of a configurator application

In the previous chapters we have dealt with attacks and defences on digital privacy in the context of web browsing. We have selected methods to mitigate these threats. By conducting a survey we could confirm our assumption, that there is a gap between desired user privacy and actual browser configurations.
In the following we will design and implement an application, that is intended to close or narrow this deviation. We will then evaluate our results and find a conclusion to our research question.

The source code, final project files and a short documentation will be hosted at:
`https://github.com/arauschecker/UBPrivacy`

4.1. Requirements analysis

Firstly, we want to revise the knowledge and understanding, that we have acquired until now and come up with a set of requirements, that forms the foundation for further decisions towards a working application.

4.1.1. User model

The average user, that we want to develop for, does not have a very deep understanding of the technicalities behind the configuration of a browser. Thus we need to create a Graphical User Interface (GUI), with clear annotations and simple mechanics.

Meetings with BHC Laboratory OÜ (cyber security consulting & solutions) have shown, that there is an additional interest in using the program to configure multiple machines automatically. Therefore we want to provide an additional "headless" mode, where the configuration will be installed non-graphically and without user interaction.

4.1.2. Operating system compatibility

The next important point we need to consider, is operating system compatibility. All major web browsers run on the three platforms Windows, macOS and Linux/Unix, which we should optimally be doing as well.

Since GUIs usually require OS dependent libraries, the best option lies in choosing JAVA as a programming language. The graphical components will run on all three system types and specific OS adaptations can be made within the classes (e.g. file path differences). We will end up with a portable .JAR executable, that holds all our resources. An installation will therefore not be required.

4.1.3. Browser compatibility

Similar to the previous operating system requirement, we also need to support major web browsers. While the limited time frame made us choose to configure Mozilla Firefox only, the design of the application should still account for an extension with other browsers.

4.1.4. Extensibility

A last requirement, that plays an important role in the program development is extensibility: With the ever growing amount of features in today's browsers, the attack surface expands as well. The use of clean abstraction layers and modular interfaces should ensure easy updates of our application in the future.

4.2. Application design

In the following we will combine the discussed requirements and previous defence strategies into a more detailed software model. On the basis of which the actual application will be developed.

4.2.1. Separation of concerns

Our program can be split into two distinct parts: Firstly we want to ask the user how he likes to use his browser and what privacy aspects he deems important. Secondly, we configure the browser according to the recorded answers.

The element, that connects these two processes is a common data source: The initial user survey feeds answers into our data base, the configurator section uses this information to adapt browser settings and generate a configuration.

We want to adopt this modularity in our application design: Every module becomes a single package inside our JAVA implementation, that holds all respective classes:

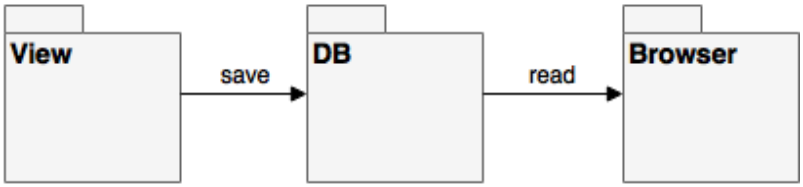

Figure 1. Package model: View, Database & Browser configurator

4.2.2. View

The view consists of three main window components, that transition to each other: An introductory screen, the user survey and a screen, that proceeds with the configuration.

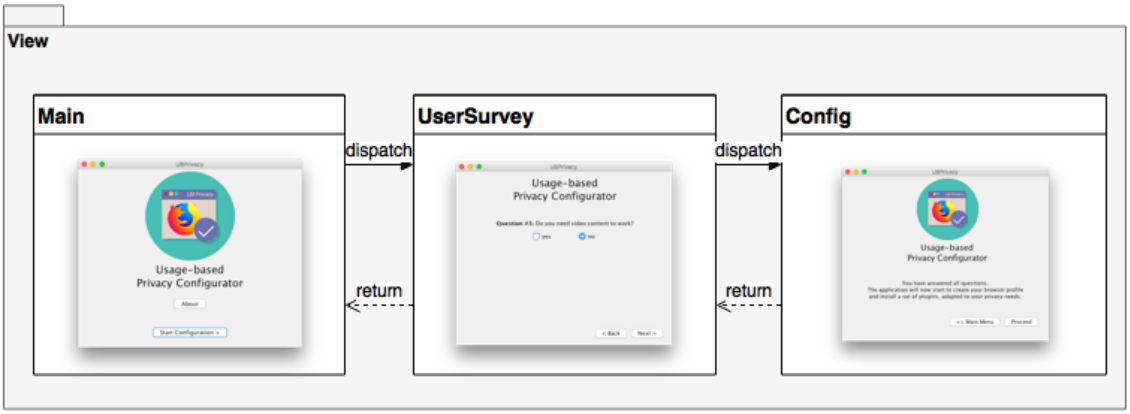

Figure 2. View - Class diagram

In order to guarantee the extensibility of the program, it seems impractical to store the survey questions inside the source code. Instead, we save them inside our database and

iterate over all rows. The label, that loads the question element supports HTML. Therefore we can add formatting and additional paragraphs, to annotate the question and make it as easy to understand, as possible.

4.2.3. Database

For the database we decided to use the Singleton pattern: the class method *getInstance()* returns a reference to the Database instance. Only one Database object can exist. Internally we use SQLite and store the entire data structure in a file. This way we do not need to start a Structured Query Language (SQL) server, thus save resources and guarantee portability.

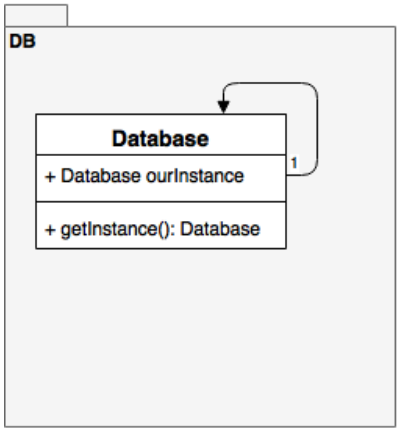

Figure 3. Database - Class diagram

The database contains two tables, to store questions and answers of the survey:

Column name	type
id	INTEGER (auto increment, primary key, not null)
question	TEXT (not null)
uid	INTEGER (default: -1, not null)

Table 2. SQL Structure of Table: SurveyQuestions

The *id* parameter is a Primary Key in the question table and a Foreign Key in the answer table. On this way we connect the answer to the question. If we want to clear all answers or modify them, we just need to operate on *SurveyAnswers*.

The parameter *uid* is an identifier, that is used to connect configurator code to a question. This will be further explained in the following section.

Column name	type
id	INTEGER (primary key, unique)
answer	VARCHAR(20)

Table 3. SQL Structure of Table: SurveyAnswers

4.2.4. Configurator

The Browser package contains the core of our application, the browser configurator. After the user has answered the initial survey, this component will adapt the browser settings.

We create a class structure, that is mirroring the functionality of the actual objects: The *Browser* class is abstract and passes on a *ConfigManager* object and a *configure()* method to its subclasses. The *Browser* class itself can hold browser specific information, such as file destinations. It represents the actual browser and offers to be configured with the respective method.

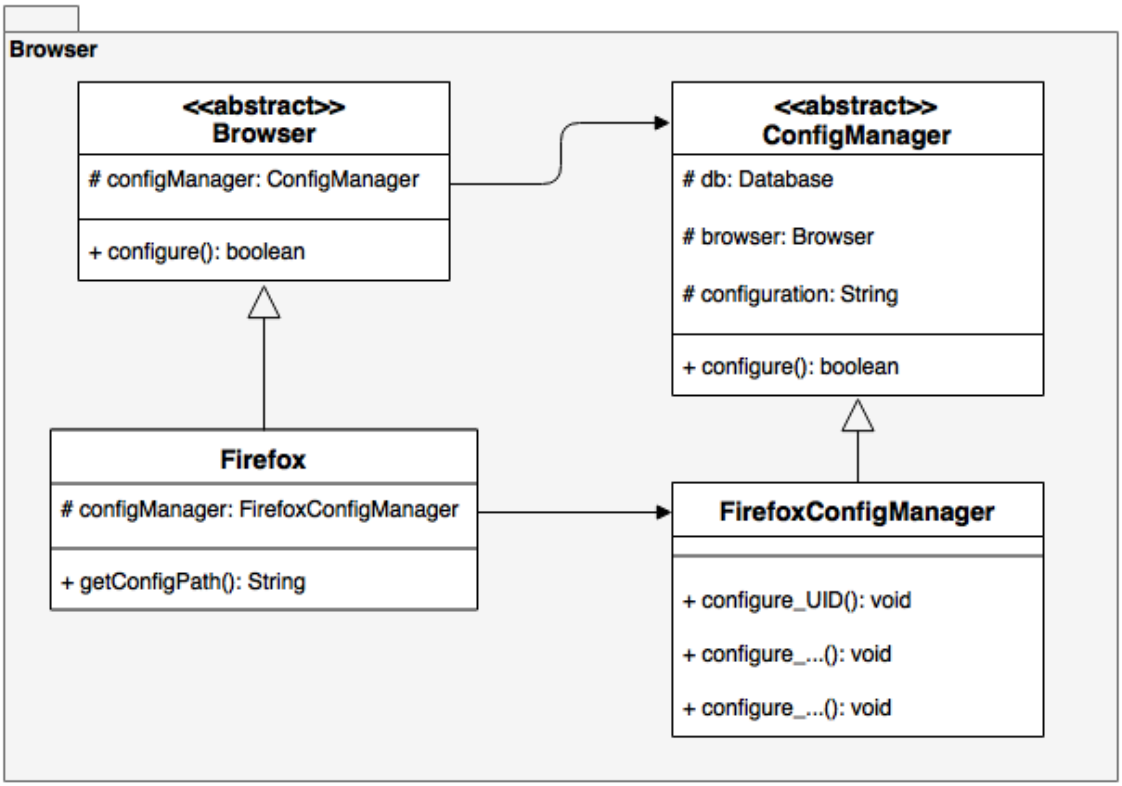

Figure 4. Configurator - Class diagram

The *ConfigManager* class is able to access the database, as well as its callee, the browser. The *configure()* method imports the answers from the database and connects the questions to configuration code.

```
List<Integer> uids = getUIDsFromDB();
java.lang.reflect.Method method;

  for(Integer uid : uids){
    try{
      String answer = getAnswerFromDB(uid);
        method = this.getClass().getMethod("configure_"+uid,
           String.class);
        method.invoke(this, answer);
    } catch (NoSuchMethodException e) {
      // No method... take the next uid.
    } catch (IllegalAccessException e) {
      e.printStackTrace();
      return false;
    } catch (InvocationTargetException e) {
      e.printStackTrace();
      return false;
    }
  }
```

Figure 5. Configurator code invocation using JAVA's Reflect API

In order to keep the actual configuration code as clean as possible, we decided to use JAVA's Reflection API to connect a question to code, using its *uid*: For all questions inside the database, we fetch their *uids* and execute existing methods, that match it. Example: Question 1 has the *uid* **11**. The *configure()* method automatically calls *configure_**11**()*, in case the method exists. This is achieved with the above code listing.

In the *FirefoxConfigManager* class we are then able to specify the configuration format and behaviour for a positive or negative question answer.

4.2.5. Question design

After describing the main design decisions behind our application, there is one last point, that remains: Reducing our several defence strategies into short and easily understandable questions and splitting the configuration keys into respective blocks.

The following questions have made the selection into the final program. Comments on our choices are appended.

1. **Do you need to use Adobe Flash?**
   As explained in section 2.4.1, we want to disable Flash to reduce the attack surface regarding exploits. However, the results of the survey we conducted show, that some participants still need to use these NPAPI plug-ins. We want to allow the user to choose himself, but also warn him about the connected risks. We note, that HTML5 technology has replaced Flash for common video playback and should only be used, if no alternatives exist.

2. **Do you need to use Microsoft Silverlight?**
   This plug-in is, similarly to Flash, also used by some users. We want to apply the same policy as to the previous question.

3. **Do you want to block dangerous websites?**
   Positively answering this question will activate the content filtering, that is a part of the configuration subset of section 2.4.1: *browser.safebrowsing* components filter malware and phishing sites and alert the user upon visit.

4. **Should we disable experimental and possibly unsafe features?**
   This question will toggle the remaining restrictions we compiled in section 2.4.1. Unnecessary, experimental functionality will be shut off.

5. **Do you want to reduce your browser fingerprint?**
   This question turns on our anti-fingerprinting mechanisms. We explain shortly, how fingerprinting allows Internet services to recognize a system and also explain, that this will only reduce the fingerprint, as a full defence is practically impossible.

6. **Do you want to block common advertisements?**
   This item will block basic advertisers as a side effect of the tracking protection, described in section 2.4.3. We will enable only the common, instead of the *strict* block list.

7. **Do you want to use enhanced tracking protection?**
   Positively answering this question will enable the advanced tracking protection, that uses the *strict* blocking list. We will set additional parameters, that we discussed in section 2.4.3, like cookie lifetime and behaviour, as well as disk caches and DNS caching.

8. **Do you want to remove personal data upon browser shutdown?**
   This last question will add the *privacy.clearOnShutdown* configuration components to the browser, which clears all personal information, when the browser is closed.

4.3. Testing

We used the previous sections to discuss the requirements of our application and implemented a tool, based on our design decisions. Now we want to verify, that all components work as expected and analyse results on how user privacy is affected in different configuration scenarios.

4.3.1. Exploit risk

Here we want to check, if plug-ins are correctly deactivated when question one and two are negated. Furthermore we want to verify, if additional components are disabled, when question four is answered positively. (questions: see section 4.2.5)

For question one (regarding Adobe Flash), we include the configuration parameter *plugin.state.flash* $\rightarrow$ *0*, if the user does not signal the need to use the plug-in.

Upon starting the browser we are presented with an error message by Firefox:

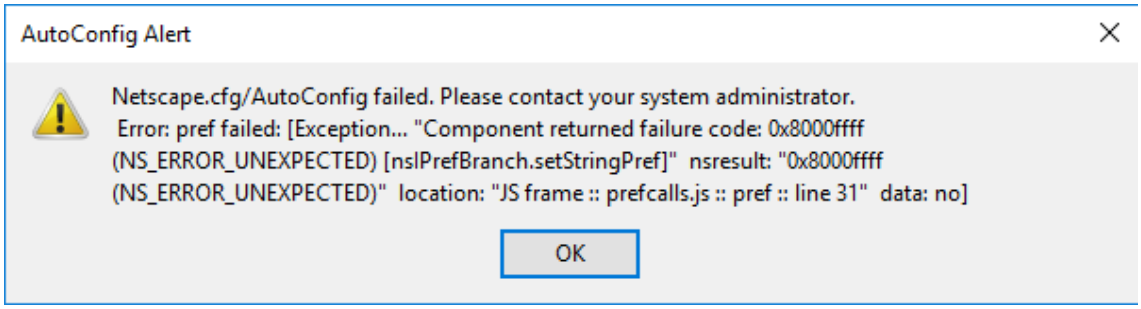

Figure 6. Firefox start-up - Configuration loading error

Further investigating the cause, we find, that Firefox could not load the parameter because we accidentally enclosed the value in double quotes. Removing these and directly setting *pref("plugin.state.flash", 0);* works without problems.

Retrying our configuration, the browser start-up now succeeds. Checking Firefox's internal status page *about:plugins*, we can see, that Flash is now deactivated:

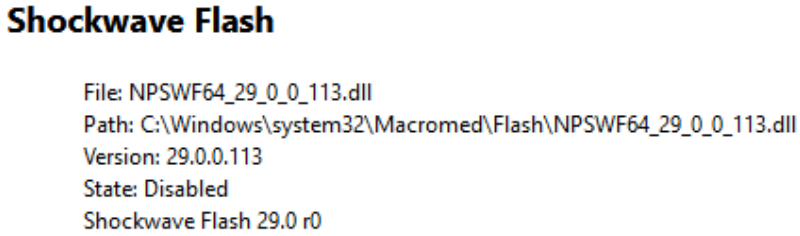

Figure 7. Firefox post-configuration plugin overview - Flash disabled

As a next step we check Firefox's internal *about:support* page, that shows us, which configuration parameters were loaded. We can verify, that all our custom items were successfully loaded.

4.3.2. Fingerprinting

Regarding our anti-fingerprinting measures we can obtain from Firefox's *about:support* page, that our parameters have been loaded. However, checking some of the fingerprinting attributes of section 2.4.2, we can see, that the adaptations, that Firefox's *privacy.resist.fingerprinting* mode is supposed to implement, have not changed (e.g. the time zone is set to our regional value and HTML5 canvases are still loaded).

Investigating this issue, we see, that the parameter *privacy.resist.fingerprinting* is overwritten by *privacy.resistFingerprinting*:

Preference Name	Status	Type	Value
privacy.resist.fingerprinting	**modified**	**boolean**	**true**
privacy.resistFingerprinting	default	boolean	false

Figure 8. Firefox fingerprinting parameter overload issue

On Mozilla's support pages we can find both parameter names used [32], however, in our understanding, *resist.fingerprinting* appears to be deprecated and *resistFingerprinting* preferred. To fix our overloading issue, we thus update our application to use only the latter in the configurator.

The fingerprinting protections are now correctly activated, so we want to analyse, how our adapted configuration compares to default settings. For this evaluation the Panopticlick project of the Electronic Frontier Foundation is a good choice [37]: They fingerprint the

same parameters, that we selected in our defence and have collected around 1.4 million fingerprints at the time of writing this thesis.

We execute the online fingerprinting test found at *https://panopticlick.eff.org* [37] with the default installation configuration of Firefox to generate the pre-defence results, then use our application to install a protective configuration and test again, to generate the post-defence values:

Panopticlick displays the Shannon's entropy per attribute and also the dataset size, which allows us to calculate the, data size independent, normalized entropy, as we explained in section 2.3.2.4. The dataset size was ~1,400,500, which yields a maximum entropy of 20.42 bits.

Attribute	**Norm. Ent. (Entropy) Pre**	**Norm. Ent. (Entropy) Post**
List of plug-ins	0.42 (8.59)	0.06 (1.25)
User agent	0.35 (7.17)	0.29 (5.58)
List of fonts (JS)	0.29 (5.84)	0.36 (7.40)
HTML5 Canvas	0.48 (9.72)	0.21 (4.24)
Content language	0.21 (4.25)	0.05 (0.92)
Screen resolution (JS)	0.12 (2.45)	0.44 (9.07)
HTTP Accept header	0.26 (5.29)	0.10 (2.02)
WebGL hash	0.65 (13.23)	0.12 (2.48)
Timezone	0.15 (3.16)	0.15 (3.16)

Table 4. Fingerprinting results (normalized Shannon's entropy) pre and post defence (Windows 10)

As we can see, the list of plug-ins, user agent, HTML5 canvas, content language, HTTP Accept header and WebGL hash have greatly improved in their entropy. However, we can also observe significant degradations regarding list of fonts and screen resolution entropy.

For the list of fonts, the reason can be directly found in the value, that Panopticlick reports (see Appendix B): The fingerprinting defence removes *Arial Black* from the list of fonts. While Mozilla's defence strategy helps to eliminate custom fonts in the listing, it seems, that this font should be kept in the list.

The increased screen resolution entropy is a far more complex problem, if we look into the reasons behind Mozilla's defence: A website can use the *screen.availWidth / .availHeight* Javascript properties to determine the screen dimensions. However, it is important to note, that this returns the dimensions *minus* interface features like taskbars [38]. This means, that an attacker can easily identify systems with custom interface elements (espe-

cially Linux distributions with custom interfaces). In order to reduce this hidden entropy, Mozilla sets window resolution = screen resolution. This way the attacker can not extract this kind of information. Spoofing the JS values is not a valid alternative, since the screen dimensions can be double checked via CSS min and max properties.

While the handling of this multifaceted problem is still discussed and "needs_revision" on the bug report boards of Mozilla [39], we need to accept the overall degradation this brings for Windows and macOS systems, even if they commonly do not use custom interfaces.

We opened a support request on the Firefox forums to create an option to turn off the screen spoofing behaviour.

4.3.2.1. Conclusion Looking at the fingerprinting defence results we can say, that we have made overall improvements in entropy. The majority of privacy plug-ins and tools, that are used (e.g. AdBlock, uBlock Origin, Ghostery) work using blacklists and do not implement fingerprinting protections at all. Only the TorBrowser [3] takes extensive measures against this form of identity tracking. Thus, our configuration is a first step towards making it harder to track users on the Internet. However, it is important to note, that we focused on a limited set of attributes and some of them could not be defended out of current limitations in the browser's implementation. Additionally, pages like Panopticlick do not use all fingerprinting techniques, that are available (e.g. audio fingerprinting). This means, that, despite our efforts, it will still be possible to fingerprint the browser, based on a wider set of attributes.

In our opinion, the protection is strongly dependent on the average defence efforts: As long as the majority of Internet users does not use fingerprinting protections, the advertisement companies have no need to broaden their fingerprinting techniques.

Another interesting perspective to analyse, is long-term fingerprint characteristics: Attributes like window size give good entropy to identify a browser during one session, however, upon restart, the exact same window size is not guaranteed. Also within a session, the user might resize the window. While defence against all fingerprinting techniques is a continuous challenge, a focus on long-termed defence has more potential: Looking at the attributes, that return a constant value, there is clearly fewer items. By blocking all these methods (e.g. HTML5 canvas hash, WebGL rendering hash), it becomes considerably harder to identify a browser and link sessions.

4.3.3. Tracking

As we showed in section 2.3.3.4, cookie tracking is the most prevalent form of tracking. Secondly, it is practically impossible to measure tracking defence for tracking techniques like session identifier transmission. Thus we want to compare our defence efforts assessing cookie tracking improvements only. In order to do so, we want to utilize *Lightbeam*, a tool by Mozilla, that lists third-party cookies, while browsing the Internet [40].
To simulate common user behaviour, we want to visit the 30 most visited sites in the United States, according to the Alexa rankings [41]. We filter websites, that contain adult content. A full list of the pages we use, can be found in Appendix C. We visit each page and wait until it has fully loaded. Then we continue with the next one.

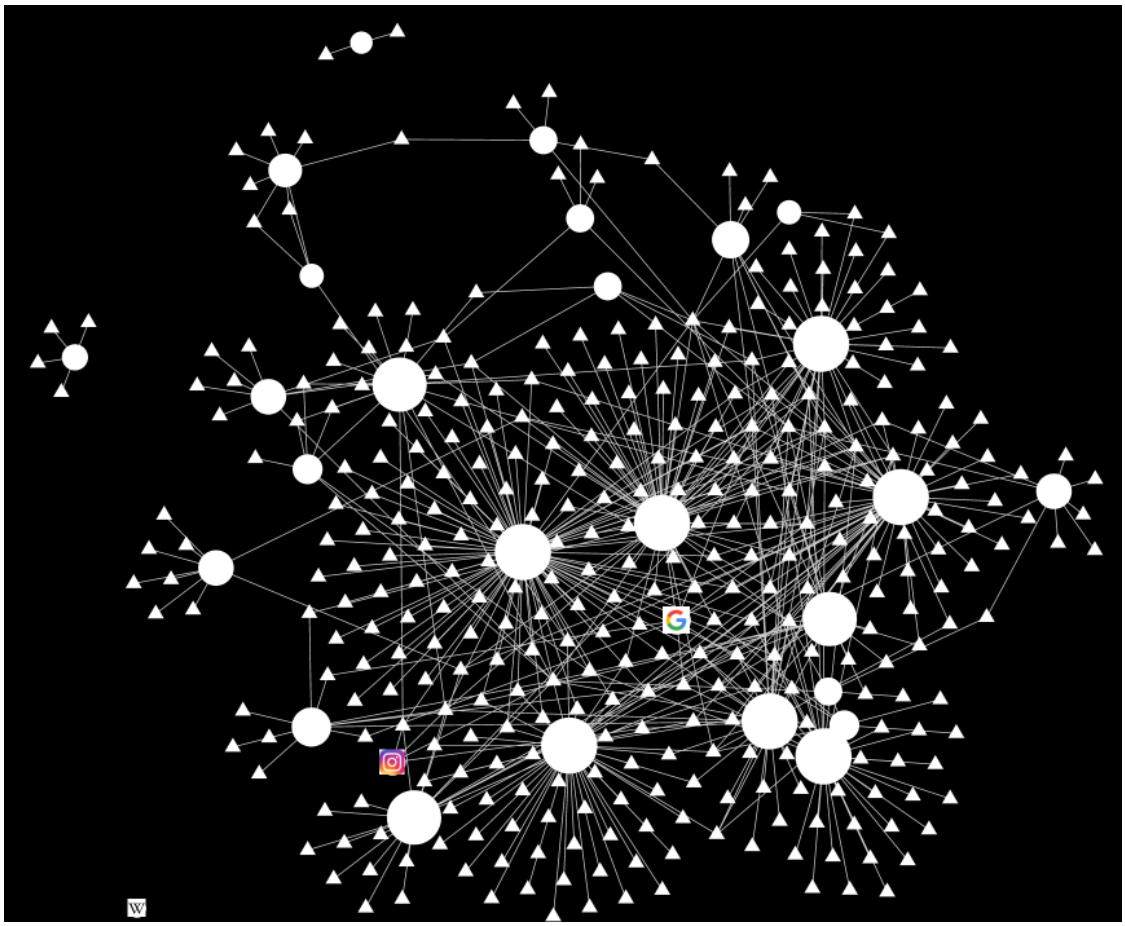

Figure 9. Mozilla Lightbeam graph - Firefox default configuration

For the default configuration, Lightbeam displays 350 third party entities. The graph shows our visited sites as circles and the third parties as triangles. The main observation, that we can make, is, how well interconnected the pages are. This means, that a big number of third party cookie domains were shared by different sites.

As explained in section 2.3.3, these third parties are able to exchange user browsing behaviour and now know, which pages have been visited and in which order, also what content has possibly been clicked.

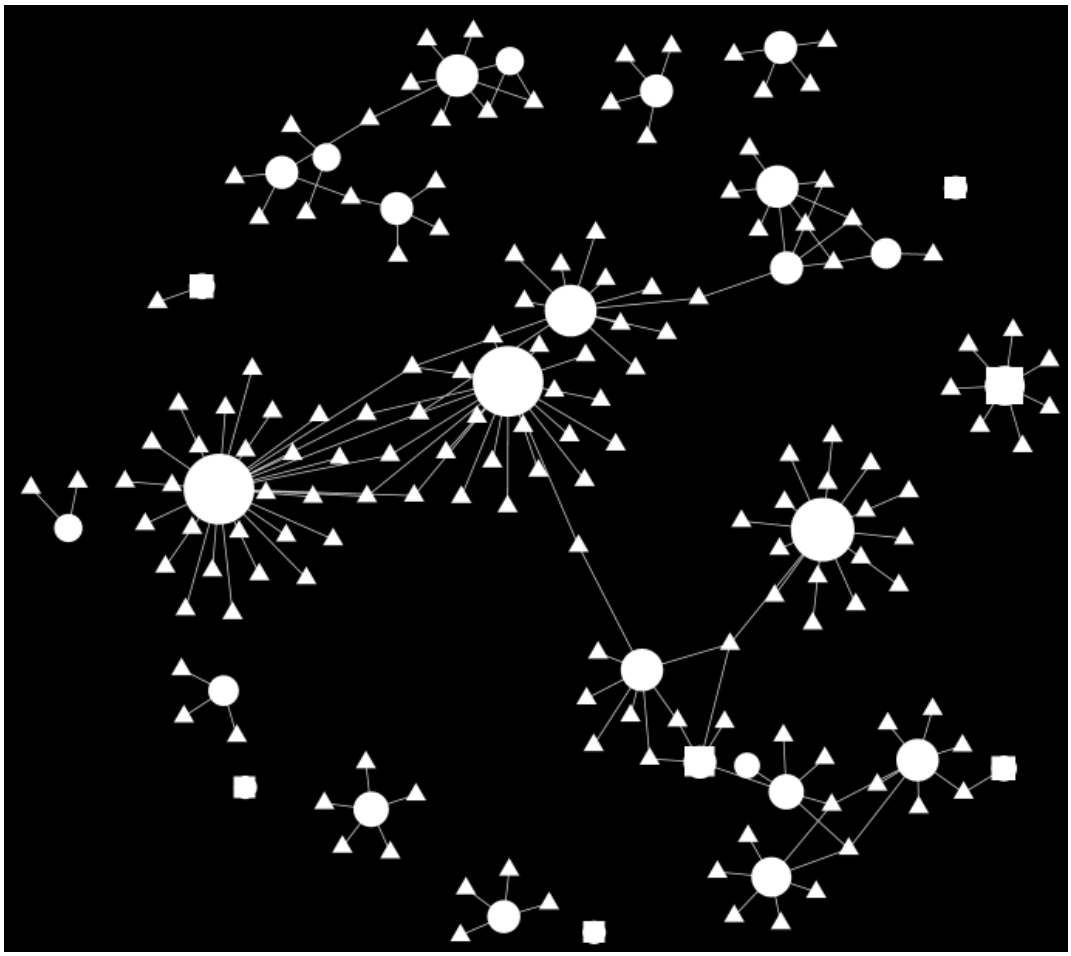

Figure 10. Mozilla Lightbeam graph - Firefox full privacy configuration

After we run our application and choose the highest privacy settings, a second series of page visits yields 150 third party entries. This is a more than 50% decrease in third party cookies. Evaluating the new graph, we can also see, that the dense mesh reduced itself to considerably fewer connections. We can see several pages, that are individual, cut-off entities, which means, that no cookie data is shared with others. (This does not mean, that the sites did not exchange user data on different ways, though. Only cookie domain shares can be tracked with Lightbeam.)

Conducting another tracking test on Panopticlick's website, we can see, that Firefox's own tracking protection does not block advertisers of the "acceptable ads" program, which means, that the above results can be improved even more by installing a stricter filtering plug-in.

4.3.3.1. Conclusion Summarizing these measurements, it can be said, that the black-listing approach visibly reduces the amount of trackers and especially the connectivity between them. Firefox's blacklist creates a good foundation for tracking defence, while even stricter lists could enhance the degree of protection. Another interesting option to increase the efficiency of tracker blocking could be a machine learning approach: Li, et al. identify, that the vast majority of trackers use long cookie lifetimes and also the value length differs from non-tracking cookies [42, Fig. 1]. In their publication, they are able to identify tracking cookies with 99.4% precision [42, section 8].

4.3.4. Headless mode

As mentioned in section 4.1.1, system administrators had interest in using our tool to configure browsers in an automated way. Therefore we implemented a headless mode, that will directly trigger the configuration routine of our application and skip the execution of the GUI.

We used the Apache Commons CLI [43] to create an extensible command-line interface:

```
[fr1dge:UBPrivacy_jar arauschecker$ java -jar UBPrivacy.jar -h
usage: Main
 -c,--config   headless configuration, with existing answers in the
               database
 -h,--help     show help
```

Figure 11. Headless mode of our application - usage overview

Using the *-c* switch, the program executes its configuration routine, provided, that the database contains answers for all questions. Otherwise the configuration is aborted.

```
[fr1dge:UBPrivacy_jar arauschecker$ java -jar UBPrivacy.jar -c
Connected to SQLite Database.
The configuration has successfully been installed!
```

Figure 12. Headless mode of our application - automatic configuration

4.3.5. Usability vs privacy

Apart from solely testing the correct functioning of our application and comparing values on test pages, it is necessary to assess, how our privacy adaptations impact usability on

the most common websites. Therefore, we want to re-use the Alexa top 30 selection, that we have made and evaluate, which sites show errors in content rendering. We visit every site and test the main functionalities it has to offer.

- **No issues found** (multimedia contents, login, main functionality working): google.com, youtube.com, facebook.com, reddit.com, amazon.com, wikipedia.org, yahoo.com, twitter.com, ebay.com, linkedin.com, imgur.com, chase.com, office.com (full online office suite was tested), microsoftonline.com, live.com, tumblr.com, craigslist.org, bing.com, paypal.com, nytimes.com, pinterest.com, microsoft.com
- *instagram.com*: The page is blank and does not render. Looking at the Javascript debugger, we can see *TypeError: message: "T.a.getLocalStorage(...) is null"*. This hints to Instagram requiring DOM storage to work. Changing the configuration parameter *dom.storage.enable* $\rightarrow$ **true** fixes the rendering issues.
- *twitch.tv*: Chat and navigation elements of the page work, video content is not played back. The debugger shows several errors like: *WebGL warning: Failed to create WebGL context* or *"https://s0.2mdn.net/instream/video/client.js" was blocked, because the tracking protection is active*. After testing various configuration settings, we could get the page to work by modifying *dom.storage.enabled* $\rightarrow$ **true** and *dom.workers.enabled* $\rightarrow$ **true**.
- *espn.com*: Needs *dom.storage.enabled* $\rightarrow$ **true** and *dom.workers.enabled* $\rightarrow$ **true** for video content to work.
- *diply.com*: The "home" page did not load fully. This could be fixed by enabling DOM storage with *dom.storage.enabled* $\rightarrow$ **true**.
- *wikia.com, cnn.com*: Videos do not work. Setting DOM storage fixes playback.
- *imdb.com*: Video navigator buttons do not work. Setting DOM storage fixes this.
- *netflix.com*: Was not tested, because accounts require a paid subscription.

4.3.5.1. Conclusion Summarizing the usability aspects we can say, that 22 out of 29 pages had no problems with their rendering. The main issue, that occurred was limited video content playback. This happened due to pages attempting to save player elements within the DOM storage, which failed because we had disabled it. Interestingly, we had no problems using more complex sites like the Microsoft Office suites. The problematic

websites all hosted entertainment media. Introducing another question "*Is video content a high priority for you?*" into our application, can address these problems.

4.3.6. Beta testing and user interview

After verifying these various aspects of our program, we ran a beta test, in order to better understand the application from a user perspective. We distributed the application to some of our survey participants and conducted interviews, while they were using the tool for the first time or after they had tested it themselves. In the following we will discuss some aspects, that were new or enhanced our perspective:

- **Varying technical understandings:** Wording our application questions, we attempted to keep the complexity level and technical depth as low as possible. Individuals, with basic background on browser security had no problems answering. However, especially users with no prior interest in digital privacy had difficulties to understand various terms and often wanted more examples or details on the topic. This means, that we need to either begin our application's survey with a user categorization, based on contextual knowledge or find ways of displaying additional annotations, when they are needed.

  Some users also did not know, how to answer questions, that they deemed irrelevant. Here we got the recommendation to mark and preselect preferred answers, so they can be skipped, if desired.

- **Question complexity & user interest:** Similar to the previous point, we discovered, that, the question complexity level directly correlated to the attention span and interest of the user: If the questions were too hard to understand, the participants quickly lost motivation to continue answering. We received comments, that some visual elements and a question progress bar would make the answering process less monotonous. Apart from this idea we think, that, again, user categorization could aid in matching the complexity levels more closely.

- **Trust & application design:** Some users reported, that the Java buttons in the application looked less trustworthy, as they were different to the classical Windows design. Font selection, image quality and layout were also considered important in forming the trust relation towards the application.

While most of these elements can be adapted quickly, it is interesting to see, that the overall interface design plays such a big factor in trusting the application. We did not realize the weight of this in our previous steps.

A second reported factor, was, that the configured browser did not display any metrics on the effectiveness of the defence, which made some individuals wonder, whether the program was functioning correctly. This is an important point, that needs to be treated in the future. While an own browser extension (e.g. button with counter) could be a good starting point for a solution, it is a great challenge to find a statistic, that is easily understandable by the user and sums up all defence components, that we implement.

- **Answer semantics:** Some participants indicated, that it was confusing for them, when semantics of answers changed throughout the questions: This means, that 'yes' should always allow an action or have a positive connotation, while 'no' should always block or have restrictive meaning. An easy example is the question "Do you want to block Flash?": Answering 'yes' would restrict in this case. Rewording the question to "Do you want to allow Flash?", answering 'no' would now act restrictively. This needs to be consistent throughout the application.

- **Accessibility:** A last aspect, that we found was, that some users tried to resize the window, while the font size stayed constant. We received comments, that the font size should then increase automatically, so the interface content becomes readable more easily.

4.3.6.1. Improved user interface mockup Taking these previous insights into account, we want to propose an improved user interface. The initial version of our application looks as follows:

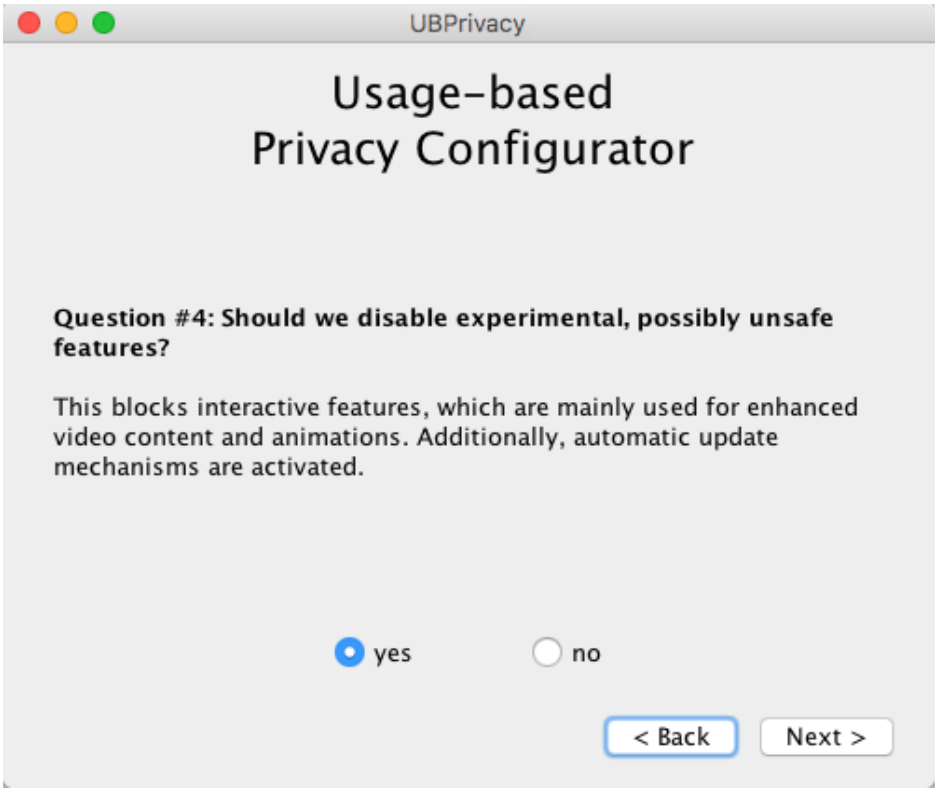

Figure 13. Our privacy configurator application - survey view

Our updated interface idea can be seen in the following:

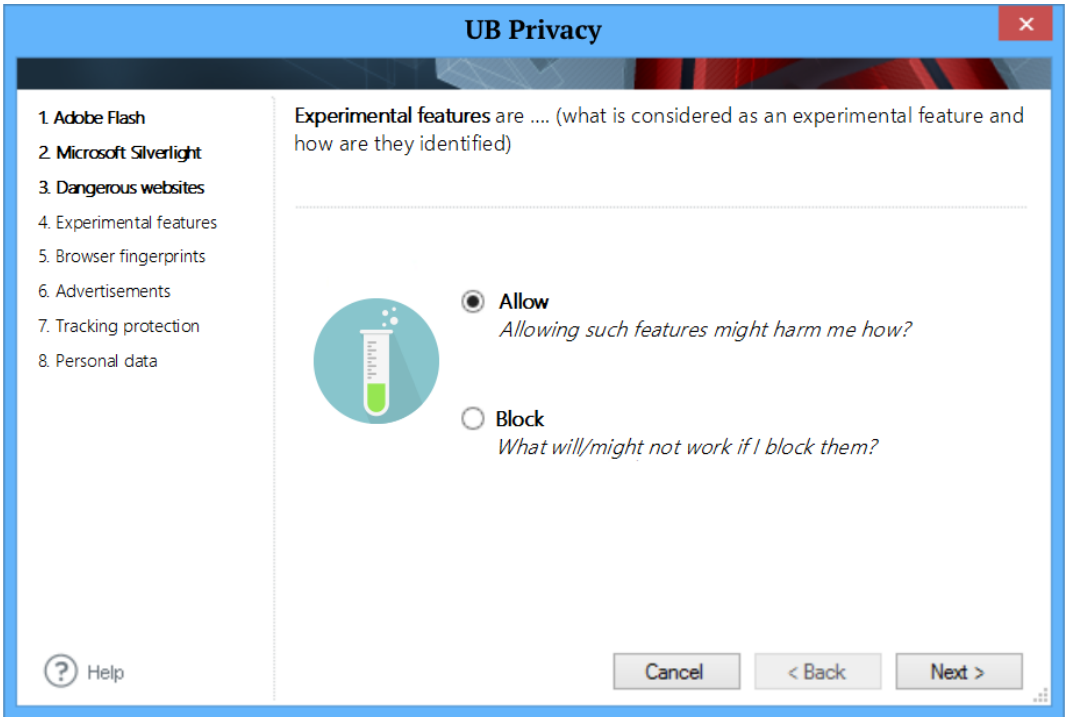

Figure 14. Proposal of an improved application interface - survey view

On the left we introduce a question overview, so the user knows how many questions remain and where he is at the moment. The main panel now includes an image, that relates to the topic, which makes the overall reception of the view less monotonic. We reworded 'yes' and 'no' to more representative answers and made space for additional annotations regarding the answers.

The limited time frame of the thesis did not allow us to implement the design update. Despite this, we wanted to give a short outline, how the look could be improved in the future.

5. Conclusion and outlook

The thesis started with an analysis of digital privacy in the context of web browsing. By discussing possible attack surfaces and their defence strategies, it became clear, that effective protection requires several configuration steps and a technical understanding of the threats. In a survey, it then evaluated, how users set up their browsers and what privacy aspects were considered relevant. While the small answer set did not allow for quantitative measurements, it was still evident, that the expected protection frequently deviated from actual browser settings. In order to enhance individual privacy, the thesis then came up with a new defence concept to narrow this gap: Instead of forcing users to adapt browser settings themselves, a configurator program asks for browsing behaviour and usage intents. Then an automated configuration takes place.
Looking at the results of the research, it can be said, that the application greatly reduced configuration complexity. Comparing default settings with our adapted configurations, we can register protection improvements in all addressed sections (exploit risk, fingerprinting & tracking). While our current adaptations certainly do not represent the best possible defence, the application allows future refinements, due to its modular design. Critical aspects of our concept were: Firstly, a relocated problem of trust: While the user initially needs to trust extensions and Internet pages to tell him how to adapt a browser to his privacy interests, he now needs to allow an external application to run and configure his device. In the survey, it became apparent, that this is problematic and some individuals are critical in their approval. Secondly, it is challenging to quantify defence efforts: In the section of fingerprinting several individual entropy improvements do not necessarily reduce the overall browser entropy, since attackers can combine attributes. In the field of anti-tracking, it is hard to detect certain forms of tracking and thus complex to verify protection effectiveness or compare defence concepts. This ultimately means, that the user is lacking a metric to understand, how well the program achieves protection, which also feeds back into the trust issue. This opens up an interesting new topic for research. Overall, our solution shows great potential and emphasizes, that the, often limited, understanding of the user needs to be addressed responsibly. It also poses as a good alternative to proprietary or commercial protection software.

A similar attempt at improved user inclusion is currently being tested by Mozilla in their "Contextual Identity Project" [44]: It extends common browser tabs to allow the user to choose, what kind of activity they will perform (e.g. banking, shopping or work). Different privacy and configuration levels are then applied. At the time of writing, the extension

did not include active filtering or defence of attacks we discussed, but instead uses a clear separation of tab contexts to achieve better protection. We think, that a combination of our configuration and application efforts with this project could result in a promising defence solution.

5.1. Future work

As it was not possible to implement all of our ideas in the limited time frame of the thesis, some particularly interesting, but still missing components are listed in this last section:

- Browser extension integration: For Firefox it was possible to realise a protection concept by modifying internal settings only. However, other browsers do not have these extensive configuration possibilities. This is why it is necessary to allow installing extensions as a part of the configurator process. They could then also be used to extend Firefox, enhancing the existent configuration parameters.
- Extending browser support: We limited our work on a defence configuration for Mozilla Firefox. In the future other browsers like Google Chrome or Microsoft Edge need to be supported.
- Quantifying privacy & protection: A privacy score or some kind of metric needs to be found, that allows the user to easily understand, how well he is protected (also, how different configurator answers influence this protection).
- Reporting function for website usability issues: In the thesis we only observed issues with video playback on some of the tested pages. Especially, if the defence system is enhanced with browser extensions, the user should be able to report usability problems to the developer, so they can be fixed.
- User interface optimization: As outlined in section 4.3.6.1, there is great potential in enhancing the trust towards the application by improving its interface design. The presentation of the questions correlates directly with a user's motivation to answer correctly and deal with the topic of digital privacy.

References

[1] G. Greenwald, E. MacAskill, and L. Poitras. (2013). Edward snowden: The whistleblower behind the nsa surveillance revelations., [Online]. Available: https://www.theguardian.com/world/2013/jun/09/edward-snowden-nsa-whistleblower-surveillance (visited on 02/12/2018).

[2] CVE Details Security Database. (2018). Top 50 products by total number of "distinct" vulnerabilities, [Online]. Available: https://www.cvedetails.com/top-50-products.php?year=0 (visited on 02/18/2018).

[3] M. Perry, E. Clark, S. Murdoch, and G. Koppen. (2018). The design and implementation of the tor browser [draft], [Online]. Available: https://torproject.org/projects/torbrowser/design/ (visited on 02/18/2018).

[4] Y. Onn, M. Geva, Y. Druckman, A. Zyssman, R. L. Timor, I. Lev, A. Maroun, T. Maron, Y. Nachmani, Y. Simsolo, *et al.*, "Privacy in the digital environment," *Haifa Center of Law & Technology*, 2005.

[5] A. Maurushat, *Disclosure of Security Vulnerabilities: Legal and Ethical Issues*. Springer Science & Business Media, 2014.

[6] C. Fuchs, "The anonymous movement in the context of liberalism and socialism.," *Interface: A Journal on Social Movements*, vol. 5, no. 2, 2013.

[7] W. Al-Saqaf, "Breaking digital firewalls: Analyzing internet censorship and circumvention in the arab world," *Örebro university*, 2014.

[8] D. S. Evans, "The online advertising industry: Economics, evolution, and privacy," *Journal of Economic Perspectives*, vol. 23, no. 3, 2009.

[9] J. R. Mayer and J. C. Mitchell, "Third-party web tracking: Policy and technology," in *Security and Privacy (SP), 2012 IEEE Symposium on*, IEEE, 2012, pp. 413–427.

[10] B. Hill. (2012). Is preventing browser fingerprinting a lost cause? W3C, [Online]. Available: https://www.w3.org/wiki/images/7/7d/Is_preventing_browser_fingerprinting_a_lost_cause.pdf (visited on 03/16/2018).

[11] R. S. Liverani and N. Freeman. (2009). Abusing firefox extensions, [Online]. Available: https://www.defcon.org/images/defcon-17/dc-17-presentations/defcon-17-roberto_liverani-nick_freeman-abusing_firefox.pdf (visited on 02/28/2018).

[12] The Mozilla Foundation. (2018). Add-ons/extension signing, [Online]. Available: https://wiki.mozilla.org/Add-ons/Extension_Signing (visited on 02/28/2018).

[13] M. Maunder. (2017). Psa:4.8 million affected by chrome extension attacks targeting site owners, [Online]. Available: https://www.wordfence.com/blog/2017/08/chrome-browser-extension-attacks/ (visited on 02/28/2018).

[14] The Mozilla Foundation. (2012). Why an outdated java plugin is so serious, [Online]. Available: https://blog.mozilla.org/security/2012/04/06/why-an-outdated-java-plugin-is-so-serious/ (visited on 02/28/2018).

[15] M. Acer and C. Jackson, "Critical vulnerability in browser security metrics," *Proceedings of W2SP*, 2010.

[16] National Institute of Standards and Technology. (2015). National vulnerability db - vulnerability metrics, [Online]. Available: https://nvd.nist.gov/vuln-metrics/cvss (visited on 03/01/2018).

[17] P. Laperdrix, W. Rudametkin, and B. Baudry, "Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints," in *IEEE Symposium on Security and Privacy (S&P 2016)*, IEEE, 2016.

[18] T. Bujlow, V. Carela-Español, J. Solé-Pareta, and P. Barlet-Ros, "Web tracking: Mechanisms, implications, and defenses," *arXiv preprint arXiv:1507.07872*, 2015.

[19] A. Barth. (2011). Http state management mechanism, Internet Engineering Task Force, [Online]. Available: https://tools.ietf.org/html/rfc6265 (visited on 02/12/2018).

[20] M. Davidov. (2012). The double-edged sword of hsts persistence and privacy, [Online]. Available: https://www.leviathansecurity.com/blog/the-double-edged-sword-of-hsts-persistence-and-privacy (visited on 03/09/2018).

[21] F. Roesner, T. Kohno, and D. Wetherall, "Detecting and defending against third-party tracking on the web," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, USENIX Association, 2012.

[22] Firefox Site Compatibility Working Group. (2016). Plug-in support has been dropped other than flash, [Online]. Available: https://www.fxsitecompat.com/en-CA/docs/2016/plug-in-support-has-been-dropped-other-than-flash/ (visited on 03/12/2018).

[23] Firefox Support Forum. (2014). Why are plugins reverting back to 'always activate' after i set them to 'ask to activate', [Online]. Available: https://support.mozilla.org/en-US/questions/1019124 (visited on 03/12/2018).

[24] Mozilla. (2015). A brief guide to mozilla preferences, [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Preferences/A_brief_guide_to_Mozilla_preferences (visited on 03/12/2018).

[25] pyllyukko. (2018). User.js - firefox configuration hardening, [Online]. Available: https://github.com/pyllyukko/user.js (visited on 03/13/2018).

[26] Mozilla. (2014). Webapi/security/webtelephony, [Online]. Available: https://wiki.mozilla.org/WebAPI/Security/WebTelephony (visited on 03/13/2018).

[27] Context Information Security. (2011). Webgl - a new dimension for browser exploitation, [Online]. Available: https://www.contextis.com/blog/webgl-a-new-dimension-for-browser-exploitation (visited on 03/13/2018).

[28] MozillaZine. (2017). Network.jar.open-unsafe-types, [Online]. Available: http://kb.mozillazine.org/Network.jar.open-unsafe-types (visited on 03/13/2018).

[29] Mozilla. (2015). Mozilla foundation security advisory 2015-29, [Online]. Available: https://www.mozilla.org/en-US/security/advisories/mfsa2015-29/ (visited on 03/13/2018).

[30] National Vulnerability Database. (2015). Cve-2015-2743 detail, [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2015-2743 (visited on 03/13/2018).

[31] B. Lassey. (2015). Bug 1186948, [Online]. Available: https://hg.mozilla.org/integration/mozilla-inbound/rev/21d15da870e8 (visited on 03/14/2018).

[32] Mozilla. (2018). Security/fingerprinting, [Online]. Available: https://wiki.mozilla.org/Security/Fingerprinting (visited on 03/13/2018).

[33] ——, (2018). Security/tracking protection, [Online]. Available: https://wiki.mozilla.org/Security/Tracking_protection (visited on 03/18/2018).

[34] ——, (2013). Cookie preferences in mozilla, [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Cookies_Preferences (visited on 03/19/2018).

[35] ——, (2010). Dom.storage.enabled, [Online]. Available: http://kb.mozillazine.org/Dom.storage.enabled (visited on 03/19/2018).

[36] Stackoverflow. (2013). Firefox invalidate dns cache [closed], [Online]. Available: https://stackoverflow.com/questions/13063496/firefox-invalidate-dns-cache (visited on 03/20/2018).

[37] Electronic Frontier Foundation. (2018). Panopticlick, [Online]. Available: https://panopticlick.eff.org (visited on 04/05/2018).

[38] W3 Schools. (2018). Screen availwidth property, [Online]. Available: https://www.w3schools.com/jsref/prop_screen_availwidth.asp (visited on 04/05/2018).

[39] Mozilla TorProject. (2012). Weird screen sizes reported by panopticlick, [Online]. Available: https://trac.torproject.org/projects/tor/ticket/4810 (visited on 04/05/2018).

[40] S. Gibbs. (2013). Mozilla's lightbeam firefox tool shows who's tracking your online movements, [Online]. Available: https://www.theguardian.com/technology/2013/oct/28/mozilla-lightbeam-tracking-privacy-cookies (visited on 04/06/2018).

[41] Alexa Internet Inc. (2018). Top sites in united states, [Online]. Available: https://www.alexa.com/topsites/countries/US (visited on 04/06/2018).

[42] T.-C. Li, H. Hang, M. Faloutsos, and P. Efstathopoulos, "Trackadvisor: Taking back browsing privacy from third-party trackers," in *International Conference on Passive and Active Network Measurement*, Springer, 2015, pp. 277–289.

[43] The Apache Software Foundation. (2017). Commons cli, [Online]. Available: https://commons.apache.org/proper/commons-cli/ (visited on 04/07/2018).

[44] Mozilla. (2017). Security/contextual identity project/containers, [Online]. Available: https://wiki.mozilla.org/Security/Contextual_Identity_Project/Containers (visited on 04/11/2018).

Appendix A Survey: Users & browser configurations

Work position	**Desired media content**	**Desired privacy components**	**Installed extensions**	**Trust for ext. app**
Sysadmin	Video, Flash, Browser Apps, Interactive Chats	Ad-Block, Anti-Tracking, Anti-Fingerprinting	uBlock Origin	very low
Technical consultant	Video, Flash, Browser Apps	Ad-Block, Anti-Crypto-Mining, Anti-Tracking, Anti-Fingerprinting	McAfee Endpoint Security Web Control, uBlock Origin, NoScript	medium
Student	Video, Browser Apps, Interactive Chats	Ad-Block, Anti-Crypto-Mining	AdBlock plus	low
Security officer	Video, Browser Apps, Interactive Chats	Ad-Block, Anti-Crypto-Mining, Anti-Tracking, Anti-Fingerprinting	Browsing protection by F-Secure	medium

Table 5. Survey results: User & browser configurations

Appendix B Panopticlick fingerprinting results & values

The dataset size during our measurements was 1,400,500 (rounded 2 digits).

Attribute	Entropy	Value
List of plug-ins	8.59	Plugin 0: Shockwave Flash; Shockwave Flash 29.0 r0; NPSWF64_29_0_0_113.dll; (Adobe Flash movie; application/x-shockwave-flash; swf) (FutureSplash movie; application/futuresplash; spl).
User agent	7.17	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:59.0) Gecko/20100101 Firefox/59.0)
List of fonts (JS)	5.84	Arial, Arial Black, Calibri, Cambria, Cambria Math, Comic Sans MS, Consolas, Courier, Courier New, Georgia, Helvetica, Impact, Lucida Console, Lucida Sans Unicode, Microsoft Sans Serif, MS Gothic, MS PGothic, MS Sans Serif, MS Serif, Palatino Linotype, Segoe Print, Segoe Script, Segoe UI, Segoe UI Light, Segoe UI Semibold, Segoe UI Symbol, Tahoma, Times, Times New Roman, Trebuchet MS, Verdana, Wingdings, Wingdings 2, Wingdings 3 (via javascript)
HTML5 Canvas	9.72	496c84b5f8e9ab3efe48e7997fc90384
Content language	4.25	de
Screen resolution (JS)	2.45	1920x1080x24
HTTP Accept header	5.29	text/html, */*; text/html, */*; q=0.01 gzip, deflate, br de,en- US;q=0.7,en;q=0.3
WebGL hash	13.23	31fa886316b233029eb0df09cd4446c1
Timezone	3.16	0

Table 6. Fingerprinting results & values - Windows 10 - Pre defence

Attribute	Entropy	Value
List of plug-ins	1.25	undefined
User agent	5.58	Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:52.0) Gecko/20100101 Firefox/52.0
List of fonts (JS)	7.40	Arial, Calibri, Cambria, Cambria Math, Comic Sans MS, Consolas, Courier, Courier New, Georgia, Helvetica, Impact, Lucida Console, Lucida Sans Unicode, Microsoft Sans Serif, MS Gothic, MS PGothic, MS Sans Serif, MS Serif, Palatino Linotype, Segoe Print, Segoe Script, Segoe UI, Segoe UI Light, Segoe UI Semibold, Segoe UI Symbol, Tahoma, Times, Times New Roman, Trebuchet MS, Verdana, Wingdings, Wingdings 2, Wingdings 3 (via javascript)
HTML5 Canvas	4.24	a273d6a847f0e2a57fa0161158f12fed
Content language	0.92	en-US
Screen resolution (JS)	9.07	800x600x24
HTTP Accept header	2.02	text/html, */*; q=0.01 gzip, deflate, br en-US,en;q=0.5
WebGL hash	2.48	00000000000000000000000000000000
Timezone	3.16	0

Table 7. Fingerprinting results & values - Windows 10 - Post defence

Attribute	Entropy	Value
List of plug-ins	1.25	undefined
User agent	11.51	Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12; rv:59.0) Gecko/20100101 Firefox/59.0
List of fonts (JS)	7.01	Andale Mono, Arial, Arial Black, Arial Hebrew, Arial Narrow, Arial Rounded MT Bold, Arial Unicode MS, Comic Sans MS, Courier, Courier New, Geneva, Georgia, Helvetica, Helvetica Neue, Impact, LUCIDA GRANDE, Microsoft Sans Serif, Monaco, MYRIAD PRO, Palatino, Tahoma, Times, Times New Roman, Trebuchet MS, Verdana, Wingdings, Wingdings 2, Wingdings 3 (via javascript)
HTML5 Canvas	10.31	ad41d887798c19615a24c636a0f6ded2
Content language	4.25	de
Screen resolution (JS)	2.45	1920x1080x24
HTTP Accept header	5.29	text/html, */*; q=0.01 gzip, deflate, br de,en-US;q=0.7,en;q=0.3
WebGL hash	10.6	fcd420b0eab7c0c17e7b6ab5963be11f
Timezone	3.16	0

Table 8. Fingerprinting results & values - macOS Sierra 10.12.6 - Pre defence

Attribute	Entropy	Value
List of plug-ins	1.25	undefined
User agent	11.03	Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:52.0) Gecko/20100101 Firefox/52.0
List of fonts (JS)	7.01	Andale Mono, Arial, Arial Black, Arial Hebrew, Arial Narrow, Arial Rounded MT Bold, Arial Unicode MS, Comic Sans MS, Courier, Courier New, Geneva, Georgia, Helvetica, Helvetica Neue, Impact, LUCIDA GRANDE, Microsoft Sans Serif, Monaco, MYRIAD PRO, Palatino, Tahoma, Times, Times New Roman, Trebuchet MS, Verdana, Wingdings, Wingdings 2, Wingdings 3 (via javascript)
HTML5 Canvas	4.24	a273d6a847f0e2a57fa0161158f12fed
Content language	0.92	en-US
Screen resolution (JS)	18.83	800x598x24
HTTP Accept header	2.02	text/html, */*; q=0.01 gzip, deflate, br en-US,en;q=0.5
WebGL hash	2.48	00000000000000000000000000000000
Timezone	3.16	0

Table 9. Fingerprinting results & values - macOS Sierra 10.12.6 - Post defence

Appendix C Alexa top 30 sites - United States

(Without pages containing adult content.)

- google.com
- youtube.com
- facebook.com
- reddit.com
- amazon.com
- wikipedia.org
- yahoo.com
- twitter.com
- ebay.com
- netflix.com
- instagram.com
- linkedin.com
- twitch.tv
- imgur.com
- espn.com
- craigslist.org
- (www.)office.com
- (login.) microsoftonline.com
- live.com
- diply.com
- wikia.com
- tumblr.com
- cnn.com
- chase.com
- bing.com
- paypal.com
- nytimes.com
- pinterest.com
- imdb.com
- microsoft.com