

Privacy Observation

Aggregation System for Reproduction of Privacy Studies

Master Thesis

Author(s):

Kast, Patrice

Publication date:

2023-11

Permanent link:

<https://doi.org/10.3929/ethz-b-000662341>

Rights / license:

In Copyright - Non-Commercial Use Permitted



Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Privacy Observatory

Aggregation System for Reproduction of Privacy Studies

Master Thesis

Patrice Michael Kast

November 28, 2023

Advisors: Prof. Dr. David Basin, Karel Kubicek, Ahmed Bouhoula
Department of Computer Science, ETH Zürich

Abstract

The widespread practices of data collection and tracking on the internet drive the business models of numerous web services, which was demonstrated by numerous prior research. Despite the emergence of privacy regulations and technologies, their impact on data collection practices remains understudied. In a continuously evolving online landscape, unlike long-term studies, one-time web measurements have several limitations in their meaningfulness.

The majority of tracking studies require collecting specific data that existing internet archiving initiatives do omit. In contrast, the Privacy Observatory introduced in this thesis orchestrates long-term, regular crawls and measurements. We reimplement five influential privacy measurement studies, and evaluate their reproducibility, by using theoretical criteria from prior work, which we show are not guaranteeing practical aspects of reproducibility. Our approach relies on containerised Docker images and standardised input/output interfaces, which not only facilitates study replication but also reveals six fundamental principles crucial for ensuring the long-term replicability of such studies by future researchers.

By reimplementing these studies on the Privacy Observatory and executing them regularly, we enable continuous observation of trends in privacy regulation compliance within an immutable execution environment, offering insights into long-term developments in internet privacy practices.

Contents

| | |
|---|-----------|
| Contents | ii |
| 1 Introduction | 1 |
| 2 Background | 5 |
| 2.1 Related work | 6 |
| 2.1.1 Reproduced studies | 7 |
| 3 Study interface | 10 |
| 3.1 Module specifications | 10 |
| 3.2 Data interface | 11 |
| 3.2.1 Input generator | 12 |
| 3.2.2 Output format | 12 |
| 3.3 Image build & storage | 14 |
| 3.4 Study deployment | 14 |
| 3.5 Interface assessment | 15 |
| 3.5.1 Study definition | 15 |
| 3.5.2 Input & Output | 15 |
| 4 Platform | 17 |
| 4.1 Components | 17 |
| 4.1.1 PostgreSQL database | 17 |
| 4.1.2 RESTful API engine | 19 |
| 4.1.3 Front-end JavaScript WebApp | 20 |
| 4.1.4 Worker | 23 |
| 4.2 Platform deployment | 26 |
| 4.2.1 Centralized observatory | 26 |
| 4.2.2 Worker | 27 |
| 4.3 Security considerations | 27 |
| 4.3.1 Adversary model | 27 |

| | | |
|----------|--|-----------|
| 4.3.2 | User authentication | 27 |
| 4.3.3 | Remote code execution | 28 |
| 5 | Empirical observations | 29 |
| 5.1 | Reproducibility issues | 29 |
| 5.1.1 | P1: Dockerfile vs. Docker image | 29 |
| 5.1.2 | P2: External dependencies | 30 |
| 5.1.3 | P3: Unstable browser binaries | 30 |
| 5.1.4 | P4: Garbage collection | 30 |
| 5.1.5 | P5: Certificate expiry | 31 |
| 5.1.6 | P6: Reimplementation guidance | 31 |
| 5.1.7 | Individual studies | 32 |
| 5.2 | Reimplementation criteria | 35 |
| 5.2.1 | Dataset | 35 |
| 5.2.2 | Experiment design (building the crawler) | 36 |
| 5.2.3 | Experiment design (experiment env.) | 37 |
| 5.2.4 | Evaluation | 37 |
| 6 | Long-term analysis | 39 |
| 6.1 | CookieBlock | 39 |
| 6.2 | OmniCrawl | 41 |
| 6.3 | AnymIP | 42 |
| 6.4 | LeakyForms | 43 |
| 7 | Discussion | 44 |
| 7.1 | Ethics | 44 |
| 7.2 | Limitations | 45 |
| 7.2.1 | Long-term stateful crawls | 45 |
| 7.2.2 | Host per worker | 45 |
| 7.2.3 | Distributed / parallel studies | 46 |
| 7.3 | Future work | 46 |
| 7.3.1 | Mobile platforms | 46 |
| 7.3.2 | (Very) Long-term analysis | 47 |
| 8 | Conclusion | 48 |
| | Bibliography | 49 |
| A | Appendix | 52 |
| A.1 | Reimplementation criteria | 52 |
| A.2 | Code base artifact | 62 |
| A.2.1 | Privacy Observatory - GitHub repository | 62 |
| A.2.2 | Reimplemented studies - DockerHub image repository | 62 |

Chapter 1

Introduction

The replication crisis is a growing concern in the scientific community, highlighting doubts about the reliability and replicability of research findings, particularly in psychology, social sciences, and biomedical research as pointed out by Moones et al. [1]. This phenomenon has sparked questions about the credibility of scientific research, and whilst it had a disruptive impact in other disciplines in 2010, computer science remained largely unaffected.

As highlighted by Cockburn et al. [2], Peter J. Denning demanded in 1980 that computer science should live up to the traditional standards of science and that replication and repeatability in the field of computer science is of significant importance. Especially in a dynamic environment like the internet, the slightest modifications in the configuration of measurement setups can have huge implications on the observed behaviour of websites, as Demir et al. [3] showed. Whilst requirements on how to enable and ensure repeatability for crawling studies were researched and discussed in the past by Demir et al. [4], these were not evaluated in practice and a common and unified approach on how to fulfil them has not yet been widely adopted.

In comparison to other disciplines, computer science leverages the use of source code to produce or generate measurements or results. Considering the subject of reproducibility, this source code often plays a central role in the efforts of reproducing past results, thus only as late as 2017, the ACSAC conference enabled artefact submission.¹ Benzel noted that article [5], conferences like the USENIX Security nowadays go as far as requiring an applicant to answer 23 detailed questions and a set of complex descriptions regarding their study execution environment. This creates an additional significant workload due to the level of detail and the large number of items required for an artefact submission and induces an overhead for assessing the artefacts.

¹<https://www.acsac.org/2017/cfp/>

Our work

In this thesis, we evaluate and address the reproducibility issues in the field of web privacy measurements. First, we developed the Privacy Observatory platform, which simplifies the maintenance of long-term studies. This platform streamlines the process of regularly repeated crawls and their respective post-processing in order to verify long-term trends over the years with a stable configuration of the study environment. Secondly, we reproduce five influential privacy measurement studies, evaluating both the suitability of our Privacy Observatory platform and the reproducibility of these studies.

The central idea of our approach focuses on the use of containerised images, in particular Docker Images. Source code that is not containerised leaves space for several factors negatively influencing reproducibility, such as variety in deployment configuration or versions of used programs and libraries. In contrast, Docker containers freeze² the software and configuration at the point of the image build. This is essential for web crawling which has external dependencies, e.g. the existence of old compatible packages in software repositories. In addition, containerisation simplifies the effort to keep a clean browsing environment for each crawl, further reducing potential sources of bias like temporary files or collected cookies which have a negative impact on the reproducibility efforts.

Whilst Docker and distributed systems like Kubernetes found wide application in the private sector, these technologies are also well understood in the academic community, even though their use is only rarely found in artefacts of web crawling studies. In 2022, first conferences like the USENIX Security started recommending the submission of Docker images as artefacts, yet not requiring it.³ However, the call for artefacts of USENIX Security 2023 does no longer contain this mentioned recommendation.⁴

²Freeze in this context means that container images contain a fixed source code and execution environment of an application, e.g. a web crawler.

³<https://www.usenix.org/conference/usenixsecurity22/call-for-artifacts>

⁴<https://www.usenix.org/conference/usenixsecurity23/call-for-artifacts>

Contributions

We propose containerisation as a simple method to improve reproducibility and illustrate the improvement of this method both theoretically and practically. We have engineered an orchestration platform, the Privacy Observatory, which facilitates the orchestration of long-term studies as well as their post-processing. Consisting of several components, which are explained in depth in this work, the complete source codes, as well as the above-mentioned reimplemented case studies are provided as artefacts.

In this work, the theoretical criteria by Demir et al. [4] are assessed, showing that the majority of them is satisfied by containerised artefacts. In addition, we practically evaluate five studies by containerising them, showing their suitability for such effort. Furthermore, we found that while some of these studies followed the principles by Demir et al., they still face severe troubles with reproducibility, which is not the case for a containerised system.

We show that source code artefacts from published studies are not designed to guarantee long-term functionality. We observe issues that arise from integrating static certificates with a fixed expiry date. Additionally, multiple assessed published artefacts contain untested source code with references to missing files. We discovered that post-processing procedures are significantly less documented, let alone automated, as the web crawler itself is. In many cases of our scoped case studies, these scripts were not publicly available and needed to be inquired by reaching out to the original authors directly.

Thesis outline

In this master thesis report, in Chapter 2, we first introduce the limitations regarding the meaningfulness of large-scale web measurements and induced hurdles with repetitive crawls performed over a long period of time. Additionally, we also define the scopes studies, that we selected for reimplementation and replication.

In Chapter 3 we describe the engineered generic study interface, which our containerised approach is based on, followed by the related usage of these standardised Docker images on the Privacy Observatory in Chapter 4. Furthermore, the individual components of the platform are highlighted and explained in detail.

We state all gathered observations regarding our reimplementation work in Chapter 5 in the form of practical reimplementation principles. The results of the performed long-term analysis on the Privacy Observatory are discussed in Chapter 6.

Finally, we discuss our work in Chapter 7, highlighting all identified risks stemming from empirical privacy research as well as all discovered limitations currently present on the Privacy Observatory platform and conclude this master thesis in Chapter 8.

Chapter 2

Background

In this chapter, we introduce some useful background information concerning the impact and significance of long-term internet studies and we discuss the present state of their artefacts. Additionally, we highlight the progress, that has already been made in the direction of well-defined study execution environments and introduce five influential publications, which we selected for reimplementation and replication.

On the internet, data collection and tracking are ubiquitous and essential for the business of many web services. Already in 2012, Roesner et al. [6] were able to detect over 500 unique trackers by crawling the Alexa 500 Top Websites published on September 19, 2011, and found that most commercial pages are tracked by multiple parties. As a response, both privacy regulations¹ and privacy enhancing technologies emerge. It remains a cat&mouse game. Data collection services circumvent these protective technologies by introducing new methods of tracking like fingerprinting-based approaches as Englehardt et al. [7] observed by analysing one million websites by leveraging the OpenWPM framework. Furthermore, Acar et al. [8] deduced that 5% of the top 100,000 websites employ canvas-based fingerprinting for persistent tracking without the use of cookies.

As the web is an ever-evolving environment, the meaningfulness of large-scale web measurements only conducted at a single point in time is limited compared to studies spanning over a larger time interval. To address this, web archiving projects such as the HTTPArchive or the Internet Archive emerged. Trevisan et al. [9, 10] used the Internet Archive to perform a long-term analysis of privacy policies over time. Whilst access to these archives is often simpler and less time-consuming than building a crawler and tracing its results over several months, their snapshot limitations prevent researchers from analysing browser states like user tracking behaviour, since often the

¹<https://gdpr-info.eu/>

old third-party tracking APIs are broken and unusable. Additionally, the coverage of the mentioned large-scale data collections focuses only on popular websites.

As an alternative data collection strategy, researchers like Hils et al. [11] perform repetitive crawls to perform measurements over a long period of time. In comparison to using large data collections, this often leads to a wide range of yet mostly unexplored hurdles, like missing long-term support and other time-induced challenges. Dabrowski et al. [12] followed this approach when comparing cookie privacy violations in 2019 using a Chrome-based crawler with the situation in 2016 using a Firefox-based one. The question arises as to whether these results are comparable at all, due to the changed study execution environment.

In this thesis, we focus on how specific approaches can overcome issues induced by long-term experiments, after giving a theoretical background in this chapter as well as defining the scope of case studies, on which we are going to measure the effectiveness of our reproducibility efforts.

2.1 Related work

Demir et al. [4] explored the theoretical requirements of web measurement studies' reproducibility by surveying 117 recent research papers to derive best practices for web-based measurement studies. A total of 18 documentation criteria were specified to ensure good reproducibility of a work. These criteria are grouped into categories and include requirements regarding the specification of the dataset (C1 - C4), the applications and programs used for running the crawler (C10), the specific crawling environment (C11 - C14), as well as how post-processing is performed in the evaluation category (C15 - C18). While Demir et al. studied the problem on a theoretical level, our work directly evaluates challenges faced by reproducing multiple measurement studies. We reuse the definitions from Demir et al. [4, Sec. 2] for the following key terms used throughout this thesis:

Repeatability means the same team using the same experimental setup re-performs a study,

Reproducibility denotes a different team using the original experimental setup, and

Replicability means different teams are using a different experimental setup to recreate the measurements.

In this work, we propose a practical implementation to address multiple of Demir et al.'s criteria. We will evaluate the efficiency of this method in Chapter 5, and comment on the completeness of these criteria.

Table 2.1: Summary of the effort needed to reimplement the publications, the reused components of the original artefacts of the publications and the status of the reimplementation (S = successful?).

| Publication | Effort | Reused Components | S |
|--------------|--------|--|---|
| CookieBlock | 16h | Crawling procedure, crawler framework & post-processing code | ✓ |
| CookieHunter | 51h | | ✗ |
| OmniCrawl | 41h | Crawling procedure, crawler framework & post-processing code | ✓ |
| AnymIP | 17h | Crawling procedure | ✓ |
| LeakyForms | 36h | Crawling procedure, crawler framework & post-processing code | ✓ |

Besides stating reimplementation criteria, Demir et al. [3] additionally researched the impact of a modified execution environment. By using five different measurement setups, they crawled a total of 1.7 million websites and built the experienced dependency tree of the loaded resources and components. Their measurements indicate a high degree of similarity considering the first-party nodes of the dependency tree, while third-party components, which are of most interest in privacy-focused studies, experienced a substantial deviation using the different crawling setups.

We assessed the compliance with the above mentioned 18 documentation criteria of four influential privacy measurement studies mentioned below. Moreover, we reproduced these studies to determine how measurements hold over time by using a modified containerised crawling setup.

2.1.1 Reproduced studies

A summary of all scoped reimplemented studies and reused components from the original artefacts is displayed in Table 2.1. We select publications of different scopes to investigate the reproducibility of a wide range of methods and to cover different crawling environments as use cases. The measurement methods range from simple Python requests to sophisticated frameworks such as OpenWPM using the Selenium library. The latter in particular has become increasingly popular in privacy measurement studies in recent years.²

Below, we present an overview of the individual studies selected for reimplementation, together with the justification why they were chosen.

²<https://scholar.google.com/scholar?q=OpenWPM>

CookieBlock

Bollinger et al. [13] (*CookieBlock*) analyse the compliance of websites with the European Union's General Data Protection Regulation (*GDPR*), which requires websites to inform users about personal data collection and request consent for cookies. The study used a Python-based crawler and some post-processing scripts to directly generate measurement statistics.

The publication was originally written by Dino Bollinger from ETH Zurich and was our first study to reimplement, since it won the best artefact reward of the USENIX 2022 conference, including all three artefact badges available, functional, and reproduced. These indicate a good quality and availability of the source code and enable an easy containerisation of the source code and the study environment. Additionally, the combination of different crawling methods, fast Python scripts for pretesting and a detailed scan using OpenWPM allowed us to practice the Docker approach with different technologies. Reimplementing OpenWPM was of particular interest, since this framework is not designed as a customisable library but instead uses a fork-and-develop approach.

Part of the mentioned study is the development of a browser extension, whose reproducibility is out of scope of this thesis.

CookieHunter

New security mechanisms like the HTTP Strict Transport Security (*HSTS*) were introduced by IETF, which can be triggered by website owners through the use of HTTP headers to enable security functionalities of browsers and to protect the session of the user. Drakonakis et al. [14] (*CookieHunter*) studied their adoption by websites during the full authentication flow when signing in or signing up to a website, to identify privacy loss caused by exposed cookies that could lead to cookie-hijacking. To observe authentication cookies, they had to automate the registration procedure using a web crawler.

We chose the CookieHunter publication since it is based on a custom Selenium crawler and not on OpenWPM as the CookieBlock. In addition, it uses stateful crawls and the source code was made accessible upon requesting it from the authors.

OmniCrawl

By conducting a large-scale analysis of websites' third-party advertising and tracking, Cassel et al. [15] (*OmniCrawl*) analysed the difference in tracking behaviours of privacy-focused browsers and their counterparts. Furthermore, they performed equal measurements on mobile versions of websites using mobile devices. All of these platforms were orchestrated by their framework OmniCrawl.

This study was chosen since it earned the Artifact Award at the PETS 2022 conference. This award should reflect code quality and documentation, which allows us to assess how easy it is to convert the artefact virtual machine into a containerised approach and publish it on the Privacy Observatory platform.

We limited the implementation scope to four different desktop browsers, i.e. Firefox, Tor, Chrome, and Brave, and omitted the mobile analysis, due to the hardware requirements.

AnymIP

Maass et al. [16] (*AnymIP*) researched the behaviour of website owners upon receiving a notification about a detected misconfiguration or security issue from researchers, which often results in mistrust, reachability issues or a perceived lack of importance. By assessing the Google Analytics tracking requests, it can be determined if the IP address of the website visitor is obfuscated or passed in plaintext to Google.

We decided, that besides containerising existing code artefacts, we also want to fully reimplement and replicate a complete study. We therefore decided to do this with a crawling environment that features a low complexity. We reimplemented the code of a crawler, which scans and analyses Google Analytics tracking requests by visiting websites.

This limited the scope of reimplementation for this case study solely on their used crawler to identify misconfigured Google Analytics website setups, which do not anonymize the IP address of website visitors.

LeakyForms

In order to examine the flexibility of the interfaces of the Privacy Observatory after its development, the study of Senol et al. [17] (*LeakyForms*) was chosen for reimplementation. By automatically filling out forms on websites and observing the network traffic, they showed that from a total of 100'000 websites, 1844 in the EU and 2950 websites in the US leaked user details present in forms before submission and without consent.

As this publication has multiple GitHub repositories³ as artefacts and extensive README files, the source code seems to follow modern development and documentation best practices. Additionally, it follows the classical two-step study procedure, with an internet crawl in the first step and a post-processing of the results in the second phase.

³<https://github.com/leaky-forms/leaky-forms>

Study interface

We promote the use of containerised images, for example, Docker images, in order to satisfy as many reimplementations criteria from Demir et al. [4] as possible and provide a fixed Docker environment for repetitive study executions. Additionally, we aim to standardise generalised command & control interfaces for the measurement containers, in order to design a compatible interface design with our Privacy Observatory platform. This allows the platform to manage different case studies with minimal customisation efforts and facilitates long-term analysis. An overview of the involved components is given in Fig. 3.1.

3.1 Module specifications

As explained in Chapter 4, the Privacy Observatory platform consists of a central part for study orchestration and job dispatching, while each study job is performed in a separate container. These study containers need to follow the required specifications highlighted in this section.

For each of the reimplemented case studies, we engineered a `_manager.py` Python script, which is responsible for preparing and setting up the execution environment, performing the crawling, as well as organising all post-processing tasks, to generate the final statistics.

After performing the complete study, the `_manager.py` script returns the individual measurements as a JSON object to the Privacy Observatory.

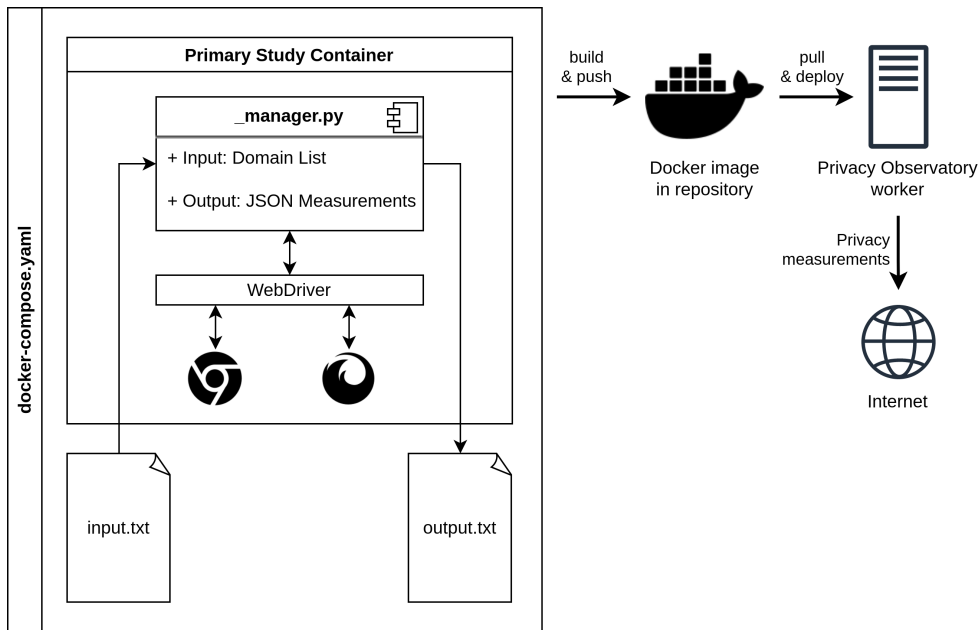


Figure 3.1: Containerised study execution environment built and pushed on a Docker image repository, as well as pulled and deployed on the Privacy Observatory worker.

3.2 Data interface

As already mentioned, the internet is a constantly changing environment and thus a fixed embedded domain-list in the Docker images may not be representational of the original scope of the studies (e.g. selecting a specific source and sample size from a website list). Therefore, a domain-list is given as input. Moreover, the output must be able to be collectable and storable for later analysis. Thus, a generic interface for input and output transfer is essential, which we model by mounting an `input.txt` and an `output.txt` file to the studies, defined in the `docker-compose.yaml` file. These interface files get mounted in the `/opt` directory of the study containers, as is visualised in Listing 3.1.

```

1 /opt/
2 - input.txt
3 - output.txt

```

Listing 3.1: Mount overview of the data interface files.

3.2.1 Input generator

Having the above-defined data interfaces to import new crawling targets to the study containers, we provide the possibility to register dynamic input generators on the platform as an alternative to static URL lists. These generators are custom Python scripts, that can generate or load the newest versions of ranked domain-lists or perform other scoping operations. An example based on the Tranco Domain List, originally created by Le Pochat et al. [18], is given in the Listing 3.2.

```

1 '\n'.join([
2     line.split(',') [1]
3     for line in
4         __import__('requests').get(
5             'https://tranco-list.eu/download/%s/1000'
6             % __import__('requests').get('https://
7         tranco-list.eu/top-1m-id').text
8         ).text.splitlines()
9 ])

```

Listing 3.2: Example domain-list Generator based on the Tranco Domain List.

The input generators are executed through the use of the `eval()` function prior to starting a new run by the RESTful API. To facilitate the deployment of the studies, these generators can be defined globally and used for multiple studies.

3.2.2 Output format

To allow a generalised output result interpretation by the Privacy Observatory platform, the following format of a JSON object was defined. It consists of two sections, where one of these sections contains aggregated key-value mappings describing the overall study statistics, and the other section contains the mappings for the individual crawled domains. An example JSON object output is shown in Listing 3.3.

The JSON file was designed to capture generic measurement types, based on our experiences from the four reimplemented studies, but we envision it to be as generic as possible. Therefore, we require mandatory fields (aggregated statistics) but we also optionally support nested structures with individual results for a specific assessed domain. Both sections of the JSON object shall use the same keys for the key-value mappings. While the aggregated statistics value type can be a number or a nested JSON object, which includes the respective mean, median, and 25/75 percentiles, only scalar measurement types can be used for the domain result values.

```
1 {
2   "stats":{
3     "wrong_purpose_wellknown":0.034482758620689655,
4     "wrong_purpose_majority":0.0,
5     "multiple_purposes":0.4482758620689655,
6     "unclassified":0.3793103448275862,
7     "undeclared":0.4827586206896552,
8     "incorrect_expiry":0.10344827586206896,
9     "implicit_consent":0.25925925925925924,
10    "ignored_choices":0.0
11  },
12  "doms":{
13    "www.google.com":{
14      "wrong_purpose_wellknown":2,
15      "multiple_purposes":7,
16      "undeclared":18,
17      "incorrect_expiry":1,
18      "wrong_purpose_majority":"None",
19      "unclassified":"None",
20      "implicit_consent":"None",
21      "ignored_choices":"None"
22    },
23    ...
24  }
25 }
```

Listing 3.3: Example output JSON object

We identify two distinctive web crawling measurement types:

Presence checks are concerned with whether a privacy violation on a website is found, and

Metering checks aim to count specific observations on a website (e.g. how many trackers were found).

While we replace the Boolean values of presence checks with their 1 / 0 counterparts for generating aggregated statistics (i.e. in the case study described in Section 2.1.1), we accept the values true / false for the key-value mappings of the individual domains.

3.3 Image build & storage

All studies must be built as Docker images to enable long-term usage of their code and the components. Rebuilding Docker images after several years may not be possible as empirically observed in Section 5.1.2, since third-party dependencies such as library repositories or specific software package versions may no longer be available. The individual requirements for the building process are explained in the below sections.

After completing the build process, the images are pushed to an image repository, which must be accessible by the worker containers, that control and orchestrate the execution of the individual studies as explained in Section 4.1.1. We propose to use the official Docker Image Repository DockerHub for images that can be open to the public and an internal image repository, such as one hosted at ETH, for images containing sensitive information, e.g. private API tokens or proprietary content.

3.4 Study deployment

To give the reader an overview of the required components, for preparing the deployment of a study on the Privacy Observatory, they are highlighted below. Also, it should illustrate the simplicity of such a deployment.

Docker images

All Docker images, that are used by the study shall be stored in an available registry, which is accessible by the Worker container. These images can be built using a `Dockerfile`, which includes the installation commands for preparing the Docker execution environment.

`docker-compose.yaml`

The specific configuration of the Docker images, environmental variables and the network configuration must be defined in a `docker-compose.yaml` file. This configuration also enables differentiating these settings among containers. This file is directly inserted into the PostgreSQL database through the Privacy Observatory.

Crontab schedule

In order to configure the requested study repetition schedule, a crontab-style definition can be made on inserting a study in the Privacy Observatory. Some example schedules are given in Listing 3.4.

```
1 weekly:          0 0 * * 0
2 monthly:        0 0 1 * *
3 quarterly:      0 0 1 */3 *
4 biannually:     0 0 1 */6 *
5 yearly:         0 0 1 1 *
```

Listing 3.4: Examples of crontab schedules for configuring study repetition.

Optional: key-value.yaml

To describe the individual key-value measurements, that are contained in the result of a study, an optional `key-value.yaml` description file can be attached to the Study Object on the Privacy Observatory.

Optional: Original measurements

The measurements from the published study can be inserted on the platform to create a reference to the situation at the time when the study was originally performed.

3.5 Interface assessment

In order to assess the compatibility of the interfaces with new studies, the LeakyForms paper was chosen after the implementation of the Privacy Observatory was complete.

As for the other papers, a `_manager.py` script was written to control the study performance process using the WebDriver of embedded browser binaries, starting by taking an `input.txt` file as well as generating a `output.txt` file after completing the post-processing.

3.5.1 Study definition

By being able to translate the source code into Docker images which are orchestrated with a `docker-compose.yaml` file, we showed that the decision to have these files as the base definition of a study on the platform was correct and fits other studies than only the originally reimplemented ones.

3.5.2 Input & Output

Similar to the original studies, the LeakyForms paper also accepted a list of domains as an input. These domains are then crawled to scan for any leaks of the user interactions. Thus, designing the input interface as a domain-list works for this crawler as well without problems.

3.5. Interface assessment

On the other hand, the output of the LeakyForms crawler after post-processing can be modelled on a `true/false` violation basis per domain, which matches the expected output format defined above.

Platform

In order to facilitate the long-term execution of web analysis studies, we engineered the Privacy Observatory platform, a framework that orchestrates the start, monitoring, and analysis of web experiments. In the following sections, we present the components of the platform and security considerations.

4.1 Components

The Privacy Observatory platform consists of the following components:

PostgreSQL Database It stores the results of the performed studies, processed on an overall per study and on a per domain statistic.

RESTful API Engine The RESTful API Engine contains the back-end logic of the Privacy Observatory platform.

Front-End JavaScript App The front-end application allows the user to configure new studies and analyse the results, as well as monitor the health of the individual workers.

Worker The worker executes studies and pushes the result back to the RESTful API.

4.1.1 PostgreSQL database

The platform is designed such that the database can be kept on the PostgreSQL cluster of ETH to ensure long-term support and operability. The database schema is automatically generated by starting the RESTful API Engine and using the SQLAlchemy ORM Library. The following relations are created.

Users

This database relation contains all users, which have access to the RESTful API as well as their hashed passwords. In addition to the standard attributes of the relation, helper functions for password verification and authentication token generation and checking are implemented as part of the `Users` object.

Workers

All registered workers are stored within this relation as well as their access tokens and their last heartbeat date. Using this date, it can be determined if the worker is still alive and responding. Individual `Studys` can be assigned specific `Workers` with customized execution environments.

Domainsets

Domainsets consist of the above-described domain-list generators, which can either be static domain-lists or dynamic using Python and are used for generating the `input.txt` files.

Studys

Studies are stored in this `Studys`¹ relation together with the corresponding `docker-compose.yaml` file, an output format and a crontab-schedule which can be used to calculate the next execution date.

Runs

Every execution of a `Study` is stored as a `Run` object, including the log output as well as the duration of the execution.

Domains

Domain objects get automatically generated once a measurement for a specific domain of a study execution needs to be stored. Together with the `WebApp`, the functionality to display the measurements of a specific domain is provided.

Measurements

Collected measurements of every run are stored in the form of key-value pairs, and optionally a foreign key to a `Domain` object. In case this foreign key is set to `null`, the corresponding measurement is an aggregated statistic over the full execution of the `Study`.

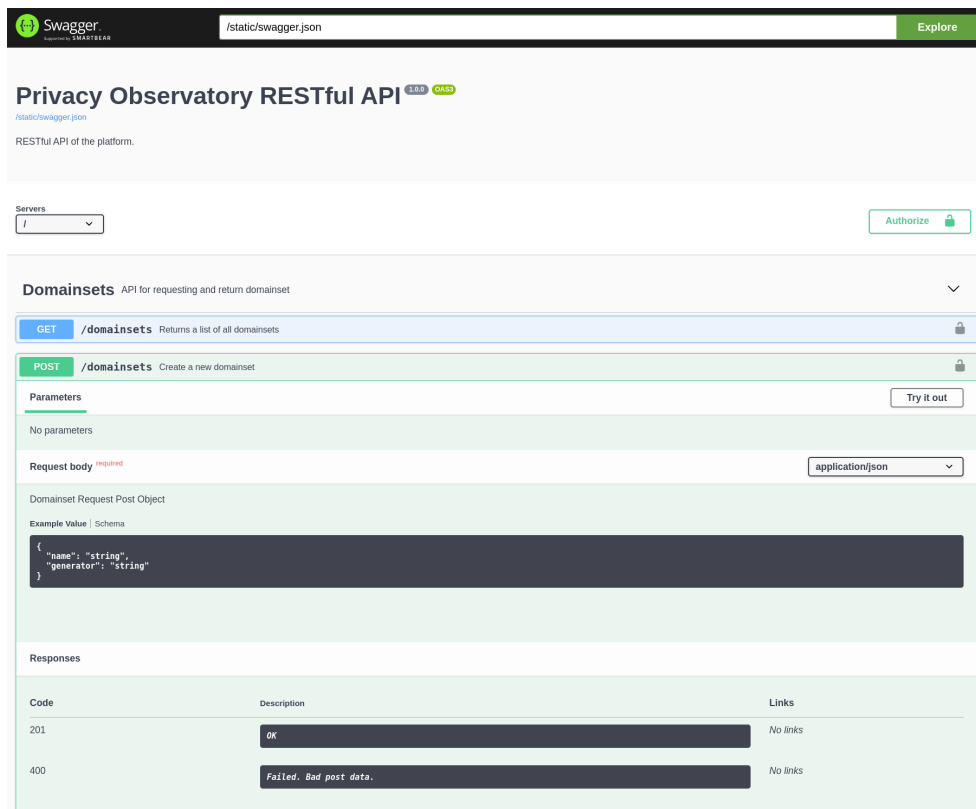
¹The misspelt term `Studys` was chosen on purpose, to avoid needing procedural platform routines to have special conjugation to access objects.

4.1.2 RESTful API engine

The RESTful API engine is based on Flask and contains the primary logic routines for the Privacy Observatory. The WebApp and the workers use the RESTful API to access the database as well as to load and store information. The individual core features of the RESTful API are explained in the following paragraphs.

Documentation

All RESTful API endpoints are documented in a static-served JSON file, which gets interpreted by Swagger and visualised on the "/docs" endpoint. Besides the visualisation that can be seen in Fig. 4.1, Swagger also provides an interactive request editor, which accelerates the development work of systems talking to the RESTful API.



The screenshot displays the Swagger UI for the Privacy Observatory RESTful API. At the top, the Swagger logo and the API title "Privacy Observatory RESTful API" are visible, along with the version "1.0.0 OKS3". The interface shows a list of endpoints, with the "POST /domainsets" endpoint selected. The selected endpoint's details are shown below, including its description "Create a new domainset", parameters (none), and request body (application/json). An example request body is shown as a JSON object: {"name": "string", "generator": "string"}. The response section shows two codes: 201 with description "OK" and 400 with description "Failed. Bad post data.".

Figure 4.1: Swagger documentation of the RESTful API.

Authentication

After a fresh initialisation of the database, no users exist in the Users relation. In order to allow the creation of Users over the RESTful API, all routes (except the JWT-generation endpoint) can be executed unauthenticated, as long as no Users exist in the database. There are no fine-grained user permissions integrated into the API endpoints for the sake of simplicity. These could be added without any problems at a later point.

HTTP Basic Authentication: After the first user is created, all subsequent RESTful API calls need to be performed authenticated by using the HTTP Basic Authentication.

JSON Web Tokens: Since storing the user password in plaintext inside of a JavaScript application is dangerous (due to cross-site scripting attacks), there exists a RESTful API endpoint to generate a JWT, which can be used to authenticate with the application instead of a username using the HTTP Basic Authentication.

Worker Token: On registering a new worker, a RESTful API access token is generated and returned, which needs to be stored for further use and passed as an environment flag to the worker. This token cannot be retrieved at a later point in time due to preventing worker impersonation.

4.1.3 Front-end JavaScript WebApp

In order to provide a simple administration on the Privacy Observatory platform, a front-end JavaScript WebApp was developed, which uses the RESTful API for processing any actions a user performs on it. Through this API-first approach, it can be ensured, that all interactions of the platform can also be accessed programmatically using the RESTful API directly. In this section, we present the views implemented in the WebApp and used for the procedural visualisation of database content.

Authentication

Authentication is handled as explained in Section 4.1.2. The user first provides his username/email and password as input for the initial login form, which can be found in Fig. 4.2. Afterwards, the WebApp fetches the JWT token and stores it in a session cookie.

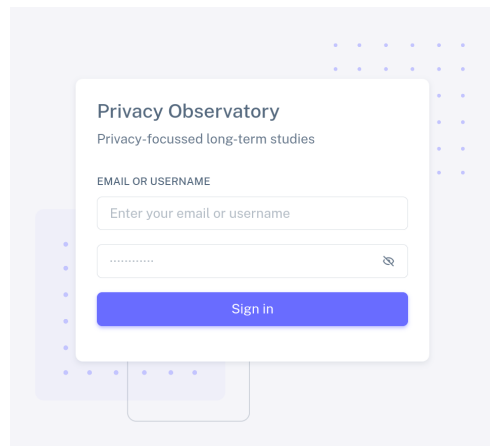


Figure 4.2: WebApp login form of the Privacy Observatory.

Domain View (Website Lookup)

Users are able to use the platform to not only review the long-term results of individual studies but also to search for specific domains and get detected violations and other measurements across multiple studies displayed, as shown in Fig. 4.3.

Object Class Views

In case the RESTful API returns an array of objects as an outer parent element, the Object Class View is rendered. An example of this view can be seen in Fig. 4.4, which visualises multiple entries of a specific database relation. The views of the different object classes are generated procedurally, with the column names for the main table automatically being extracted from the RESTful API response. This allows changing the RESTful API without the need to modify the WebApp.

A `static_settings` JavaScript object can be used to specify fine-grained properties of the attributes of the object classes. These can be descriptions or attribute configurations, like `multiline_mode` or hidden input fields, as it is done with the `Users` Object Class, where the password input field shall be displayed in the `user-edit` view but not in the `user-read` view.

Edit Modal: By clicking on the edit button on a table row, a specific entity can be edited. Similar to the procedural generation of the main table, the input fields and the labels of the edit modal are created automatically based on the RESTful API.

Besides input fields that expect a simple string value, different attribute namings modify the generation process in the following way:

4.1. Components

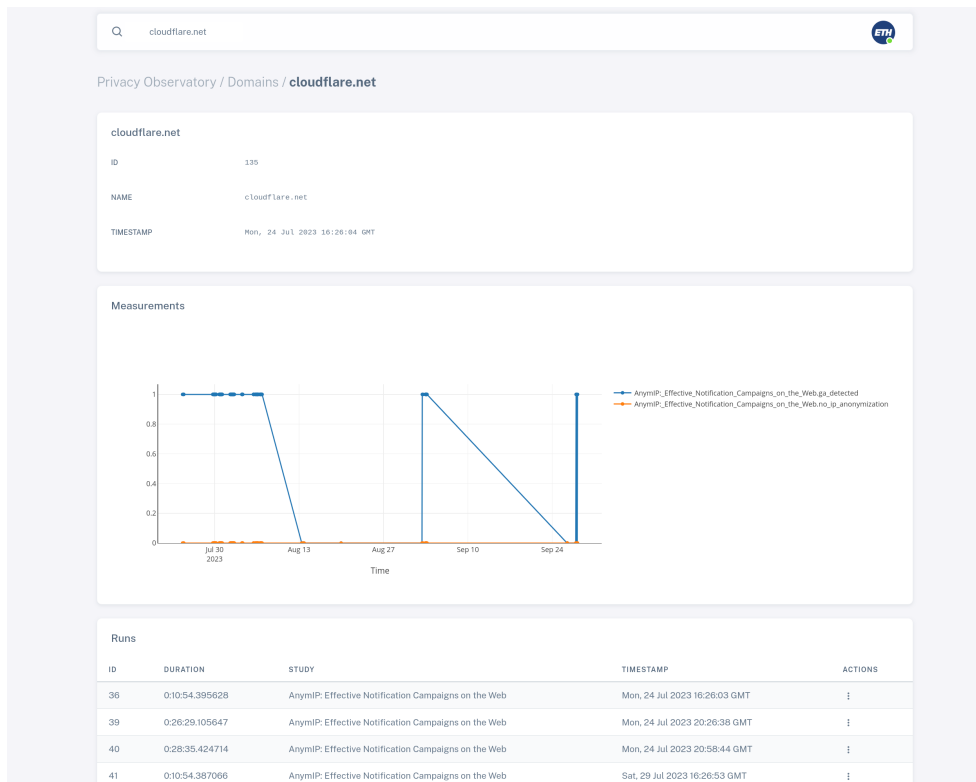


Figure 4.3: The `cloudflare.net` domain lookup on the Privacy Observatory.

1. First, it is checked if it contains an `_id`, which would indicate that it is a foreign key to another relation. If this is the case, the foreign relations database is loaded and the name attribute is displayed. In the case the attribute is not `null`, its value is used to mark the correct foreign entity as selected.
2. If the attribute name contains `num_`, this indicates that it is a counter of a relational database entity by key reference (e.g. `num_measurements`: the total number of performed measurements). Thus, this value is then marked as read-only. These numbers are also displayed on the entity page as statistic widgets instead of standard text in the card table.
3. In case the naming contains `is_`, the attribute is interpreted as a state variable set by the platform and is then marked as read-only (e.g. `is_alive`: whether or not the worker is alive).
4. As a last edge case, if the name contains `_date`, it is a date field of an entity set by the platform and thus also set to read-only (e.g. `last_scan_date`: the date on which the last scan was performed).

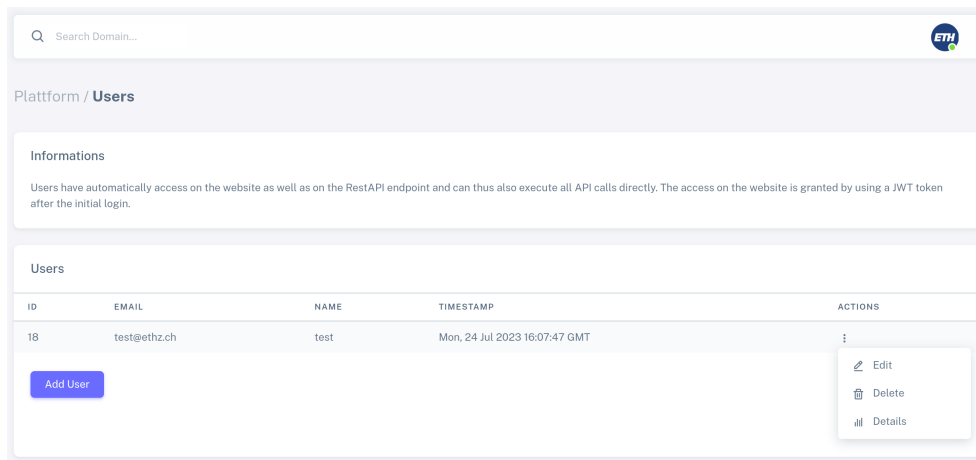


Figure 4.4: Object Class View of Users.

Object Entity View

When the RESTful API returns a dictionary as the outer parent element, the Object Entity View is rendered as shown in Fig. 4.5, which visualises a single entry of a database table as well as all linked information. Similar to the Object Class View with its Edit Modals, the Object Entity View is procedurally generated. In comparison to the Object Class View, it additionally displays widgets for attributes with the `num_` prefix and a graphical representation of results over time, rendered using the Plotly.js library for dictionary attributes, containing `datetime` elements.

4.1.4 Worker

Workers are used to control and manage the execution of studies. They use the RESTful API to get the next job to perform, trigger its associated domain-list generator, and pass its result to the study. After the study execution finishes, the results are transferred back to the RESTful API for storage.

A worker needs to send regular heartbeats for the worker itself as well as for the study it is performing. Since it needs to be able to create Docker containers for performing measurement studies, it needs to be launched on a system in the form of a privileged Docker container. By using environment flags, the worker can be configured with the hostname of the RESTful API service as well as its designated authentication token.

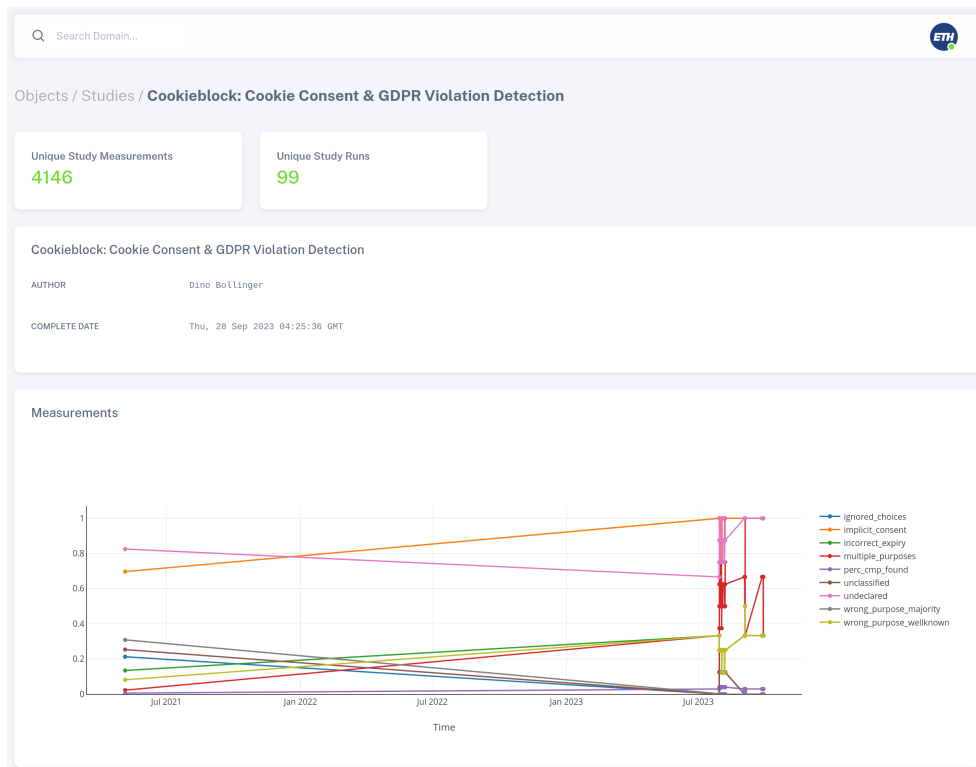


Figure 4.5: Object Entity View of a specific Study.

Study Scheduler

Designing schedulers on continuous scanning platforms poses a huge challenge regarding prioritisation and error-handling of crawling jobs, especially in the case of long-running jobs, when there can be overlaps in the execution periods (i.e. an old job takes longer than when its next execution should start).

We wanted to be able to specify the repetitive crawling interval of the studies in the crontab format. Since querying the database for the next study to perform ordered by a crontab format is not supported in PostgreSQL, we decided to cache the next execution date at the time of creation or modification of the study, when a new crontab string is supplied.

In case the worker calls the `/jobs/next` RESTful API endpoint to get a new job to execute, the API scans all studies and returns those, which have an old `heartbeat_date`, either because they completed already a while ago or because their worker was terminated. Then, it matches the found studies to find jobs, which are scheduled to run next regarding their precomputed scheduling time (based on the crontab string) or those, which did not complete yet (i.e. the old complete time is `None` or smaller than the scan time).

Handling input and output

As described in Section 3.2, the input and output of a study are exchanged through the use of two text files. Our first approach was to mount a volume map from the worker Docker container to the study Docker container, but this is not supported by Docker. The solution was to mount the `/opt/input.txt` and `/opt/output.txt` files inside the worker Docker container, in order to allow it to interact with these files. This poses a limit of one worker container per host system, since otherwise, there would be a conflict by multiple workers controlling the same input and output documents (i.e. `/opt/input.txt` and `/opt/output.txt`).

Docker-outside-of-Docker

To run Docker inside of a Docker container, we follow the Docker-outside-of-Docker approach, for which the worker container needs to run privileged and has the Docker socket of the host system mounted. The used `docker-compose.yaml` file can be seen in Listing 4.1.

```

1 version: "3.4"
2 services:
3   worker:
4     image: docker_worker
5     restart: always
6     privileged: true
7     environment:
8       # always use prefix of http:// or https://
9       api_host: http://example.com/api
10      api_tkn: XXXXXXXXXXXXXXXXXXXXXXXX
11     network_mode: "host"
12     volumes:
13       - /opt/input.txt:/opt/input.txt
14       - /opt/output.txt:/opt/output.txt
15       - /var/run/docker.sock:/var/run/docker.sock

```

Listing 4.1: Privileged `docker-compose.yaml` file of the worker

In comparison, there also exists the Docker-in-Docker approach, which can be used to spawn Docker containers, which are invisible on the host system and thus are more strictly isolated from the host system. The downside is, that a special `docker:dind` image must be used, which is a fully containerised Docker system, which does not support `docker-compose.yaml`. It also causes problems with Linux Security Modules (*LSM*) and has many other unwanted side effects.

Execution environment

Dabrowski et al. [12] as well as Urban et al. [19, 20], and Yang et al. [21] clearly showed, that different legislations like the GDPR can have a direct influence on the delivered content from a website, depending on the geolocation of the visitor. It is therefore crucial, that a specific study needs to be able to be executed by a specific worker of the Privacy Observatory, located in a predefined geolocation or having a prepared execution environment with preinstalled VPN Docker networks, to perform, for instance, GDPR-related studies from within the EU territory. The different execution environments can also be shared on one worker for different studies. We therefore provide the feature to link a study to a designated worker.

4.2 Platform deployment

There are ready to use `docker-compose.yaml.template` files for the `api`, the `webapp` and the `worker`, which just need to be renamed to `docker-compose.yaml` for local deployment. Instead of cloning the complete Git repository for each deployment, the individual containers can also be started by directly referencing the images hosted on DockerHub.

4.2.1 Centralized observatory

As a main requirement, the server on which the Privacy Observatory is running needs to be contactable by proposed future worker networks and locations. In our case with a deployment at ETH Zurich, where a centralized PostgreSQL database cluster is used, only the `docker-compose.yaml` file of the RESTful API needs to be modified accordingly.

Reverse proxy

In order to serve both the RESTful API and the WebApp HTTP endpoints on the same external IP / hostname, an Nginx reverse proxy is set up on an ETH Cloud Server, which forwards the webroot directory `/"` to the internal port 8000 and the `"/api"` to the internal port 5000 on which the RESTful API is listening. This way, any multi-port communication errors, like security features such as Cross-origin resource sharing (CORS), can be prevented. Using URL rewriting, we do not need to perform any changes on the defined routes in the source code of the RESTful API.

Also, a Web Application Firewall (WAF) is integrated into the reverse proxy using the `mod_security`² Nginx module, which offers an out-of-the-box protection for well-known exploits.

²<https://docs.nginx.com/nginx-waf/>

4.2.2 Worker

As mentioned above, the data interface files shall be mounted inside the worker container. Additionally, the user shall make sure the referenced Docker Image repositories are available without login. Otherwise, the login credentials shall be mounted inside the worker container for authentication during study orchestrating.

Using the provided `docker-compose.yaml` in Listing 4.1, the worker image can be pulled and started. Whilst the host server, on which the worker container remains, is also able to run Docker for different experiments, it is not recommended due to performance constraints as well as unwanted input and output file manipulations.

4.3 Security considerations

Since the RESTful API acts as a central component which distributes scanning jobs, several security considerations are made, which are highlighted in the following.

4.3.1 Adversary model

The RESTful API is built primarily to be hosted on the internal network of ETH. GDPR-related scanning activities require workers located at a university within the EU to be able to record privacy-driven behaviour of websites. The API has therefore been designed and hardened for internet exposure, as the API might get exposed on the external perimeter to facilitate geo-distributed study performance over different universities.

4.3.2 User authentication

The user authenticates with the web application through the use of a username/email and a password. The provided password can in theory be used for authenticating any RESTful API requests in the form of HTTP Basic Authentication, which the user has permission to perform.

To not have to store the plaintext password of the user in the client-side JavaScript of the browser, the application only uses the provided password for an initial `/users/token` RESTful API request. This then offers a JWT to the web application as a result, which shall be used for further requests. This JWT is generated based on an application secret, which can be defined through an environment variable specific to each platform deployment.

4.3.3 Remote code execution

Concerning the worker, a remote code execution risk was identified, through the injection of custom Python code as domain-list generators, which allows in theory a fully adversarial take-over of the RESTful API docker container by placing a reverse shell on it. Since only authenticated users can insert Python code, which is then executed to grep the newest domain-lists as inputs for the studies, this risk was accepted.

Empirical observations

In this chapter, we present our findings based on reproducing the selected studies. Namely, we first discuss the common issues we faced, followed by the evaluation of the reproduction process using the criteria defined by Demir et al. [4].

5.1 Reproducibility issues

We discovered several reappearing issues while reimplementing past studies in Docker. To prevent these issues, we formulated the following implementation principles (P1-P6) that need to be closely followed in future studies to facilitate long-term reproducibility.

5.1.1 P1: Dockerfile vs. Docker image

One of the design choices in Privacy Observatory design was to store the studies as either Dockerfiles or Docker images. While Dockerfiles offer more flexibility in modifying the experiment, over time the required resources might not be available anymore rendering the build impossible. Based on our reimplementing of the CookieHunter study, we found that deprecated packages are often very unstable in case they need to be reinstalled in order to reproduce a study. If Docker images are used, this problem can be mitigated completely and a more stable storage of the execution environment can be guaranteed.

We therefore recommend publishing Docker images on a public repository (such as DockerHub) and optionally also releasing the Dockerfile to enable future modifications.

5.1.2 P2: External dependencies

As we experienced with the CookieHunter, Google’s API for Gmail authentication was changed completely. Therefore, we were unable to get the old study to work without completely rewriting the old Python version 2 module of the `googleapiclient`.

We recommend designing the study framework with as few external dependencies as possible and in case email verification is needed, containerise these auxiliary services as well.

5.1.3 P3: Unstable browser binaries

We observed websites detecting the used browser version. Using old and no longer supported browser versions increases the probability of triggering bot detection mechanisms and thus blocking the crawler’s ability to scan certain websites. This limits the representativeness of studies, which are based on such web measurements. We therefore aim to upgrade the used browser binaries by the reimplemented studies to their newest counterpart.

We discovered that besides browsers’ constantly changing web-render functionalities, their API was also often unstable. These changes in the API lead to incompatibility between older libraries and new versions of the browser. Thus, an automated method of downloading the newest browser binary when starting the Docker image or building the image cannot guarantee that experiments work (in the better case) or that the measurements are not biased. We therefore had to accept the limitation of using older browser versions. Additionally, the binaries of old browser versions are sometimes unavailable, which requires the authors of the study to archive the required version themselves.

The safest possible and recommended approach to ensure reliability is to copy the browser binary directly from local storage to the Docker image and not rely on available download mirrors.

5.1.4 P4: Garbage collection

During the reimplementation of the AnymIP study, we identified a resource leak in the implementation caused by opening a new virtual display for every visited website without closing it. This leads to an error, where the maximal number of subprocesses is reached by spawning too many of these virtual display objects. Increasing the `ulimit` inside of the Docker containers would only postpone the resource drain instead of addressing the cause. We therefore fixed the code to recycle the virtual display objects for the crawls and ensured that the Chrome driver is properly stopped and recreated after a successful domain crawl.

This highlighted the importance of proper garbage collection, even if a study is implemented using a programming language with automated memory management such as Python. Thus it is recommended to test the code for any obvious memory leaks or orphan processes.

5.1.5 P5: Certificate expiry

The OmniCrawl paper revealed an unexpected problem stemming from encrypted communication between individual components of the container, with enabled certificate validation. Since these certificates often get generated during the installation of tools like mitmproxy¹, they have an expiry date, after which they get revoked and new certificates must be generated. Normally, a user would then need to import these manually into browsers like Firefox, but since Docker does not have a virtual display enabled, we only observe `[ssl_client_socket_impl.cc] handshake failed` errors.

It is recommended to disable any SSL certificate validation mechanisms taking place. In case this cannot be achieved, a repetitive certificate reimport has to be installed within the Docker container, for example in the `_manager.py` script. The commands in Listing 5.1 can be used to import new mitmproxy certificates into Chrome and Firefox.

```
1 # Chrome and OS-wide
2 curl http://mitm.it/cert/pem --output /usr/local/
   share/ca-certificates/mitmproxy-ca-cert.crt &&
   update-ca-certificates
3
4 # Firefox
5 certutil -A -n "MITM CA" -d sql:profiles/firefox86/ -
   t "C,C,C" -f "ROOT" -i ~/.mitmproxy/mitmproxy-ca.
   pem
```

Listing 5.1: Bash commands for importing a new certificate into Chrome and Firefox.

5.1.6 P6: Reimplementation guidance

To provide a measurement for reimplementation criteria C14 which is concerned with the documentation of post-processing steps, we document the communication with study authors required to reproduce their study in case the README files do not exist or the available instructions are insufficient. Namely, we report the number of emails that we needed to exchange with the authors to address the incomplete documentation or bugs observed.

¹<https://mitmproxy.org/>

Table 5.1: Observed violations of the reimplementation principles by the scoped studies. In case a principle has no observed violation marked in the table, it was detected during our reimplementation work.

| Principles | P1 | P2 | P3 | P4 | P5 | P6 |
|--------------|----|----|----|----|----|----|
| CookieBlock | | | ✗ | | | |
| CookieHunter | | ✗ | ✗ | | | |
| Omnicrawl | | | | | ✗ | ✗ |
| AnymIP | | | ✗ | | | |
| LeakyForms | | | | | | ✗ |

We required no communication with the original authors of AnymIP since we have reimplemented the study from scratch. The communication with the CookieBlock authors is also not quantified, as one of the authors is the supervisor of this thesis, such that he was addressing observed issues directly. We needed to exchange a total of five emails with the authors from OmniCrawl, to get the post-processing scripts running. While we got in touch with the author of the LeakyForms paper regarding missing post-processing documentation and obvious errors in the code, their insights did not provide the resolution we were seeking.

It is recommended to not neglect the importance of documenting the post-processing steps to enable reproducibility of the results.

5.1.7 Individual studies

In the following sections, we dive into the individual problems that we experienced during the reimplementation process of the selected case studies. An additional overview of the violated principles by the papers can be found in Table 5.1.

CookieBlock

The first study we reimplemented, CookieBlock, had a clean and nice source code as an artefact. The containerisation efforts were straightforward and the available post-processing scripts gave us the idea of a centralized `_manager.py` file for performing the full processes. Originally we planned to perform the pipeline stages of a study performed by the Privacy Observatory following the microservice principle, using a centralised controlling `_manager.py` component, such as complex data processing could fit into one Docker container. Using our approach, the full study can be performed using one Docker image instead of raising the complexity by introducing pipelined processing steps.

CookieHunter

In contrast to CookieBlock, the CookieHunter source code artefact contains scripts in both Python versions 2 and 3. Furthermore, they employ an old and deprecated `python2-google-api` for the interactions with Google as an identity provider (Single-Sign-On, SSO). Since the successor of this deprecated plugin completely changed the interface, many code segments of the CookieHunter crawler needed to be rewritten. This circumstance together with other deprecated software packages and interference between scripts in various versions of Python led to an abortion of the reimplementations.

OmniCrawl

The OmniCrawl study consists of 22 non-virtual machines, including mobile device platforms. However, the demonstration virtual machine, which was published as part of the artefact, only contained the environment for Chrome and Firefox browsers.

We thus decided to containerise this virtual machine and further extend the functionality by the respective interface-compatible Brave and Tor Browser binaries.

After reverse engineering the missing and undocumented shared library objects that were missing on the Vanilla Ubuntu 18.04 and the undocumented NodeJS version that we could determine by extracting it from the GitHub repository history, it required extensive efforts to figure out that the `[ssl_client_socket_impl.cc]` handshake failed error was originating from expired mitmproxy certificates as explained in Section 5.1.5.

The source code of the crawler, which received the Artifact Award at the PETS 2022 conference consisted of a combination of Java and Python scripts. Therefore, as the post-processing and analysis scripts were not part of the original artefact, the award did not take their quality into account. The processing was separated into multiple code snippets that required manual intermediate steps rather than a full batch script. We only got our hands on them after contacting the original authors and receiving further support from them to get the scripts working.

AnymIP

As explained in Section 2.1.1, we completely wrote the source code of the reimplementation of the AnymIP paper. In contrast to the original study, we used selenium-wire combined with a virtual Xvfb display instead of the Chrome DevTools to capture the performed requests and search for the specific patterns identifying a misconfiguration of Google Analytics and therefore a privacy violation. Even though the Chrome DevTools provide an improved bot detection evasion,² we decided to use the selenium-wire library due to multiple known bugs with the Chrome DevTools inside Docker at the time the paper was reimplemented.

LeakyForms

As described in Section 3.5, the LeakyForms reimplementation was used to validate the interfaces of the Privacy Observatory after the development of the platform was finished. Based on the publicly available source code and the well-documented GitHub repositories, we estimated the reimplementation efforts to be feasible.

We observed that the source code from the LeakyForms repositories was never tested, as multiple npm packages were missing when following the installation guide, which prevented the crawler from working and resulted in errors. Fortunately, other GitHub users reported these issues and found the missing npm `install error-stack-parser` package as they explained in an unanswered GitHub issue,³ which is over 18 months old by the time of writing this thesis. This demonstrates that many crawling code bases are completely abandoned after a study has been completed.

We observed multiple further issues. First, the installation guide missed multiple shared library objects. Second, the Chrome browser was configured to run without sandboxing, which forbids execution by the root user. Thus, the researchers executed their scripts in user space. To fix that, the source code had to be extended with an additional argument when assembling the run-time arguments passed to the framework puppeteer handling the browser. Third, the Python packages for the post-processing were missing in the `requirements.txt` file for easy installation and needed to be reverse-engineered. Fourth, the code referenced files, such as `categories.dict.pickle`, which were missing completely. This demonstrates that the source code artefacts were not tested once.

Executing the LeakyForms study on our platform required rewriting the entire leak-detection algorithm contained in the post-processing analysis.

²<https://www.zenrows.com/blog/selenium-avoid-bot-detection>

³<https://github.com/leaky-forms/leaky-forms-crawler/issues/1>

5.2 Reimplementation criteria

As introduced in Section 2.1, one aim of this study was to determine the positive effects of publishing a study on the Privacy Observatory using the containerisation approach. The containers in combination with the usage of the Privacy Observatory implicitly fulfil most of the reimplementation criteria as stated by Demir et al. [4], without any additional documentation or efforts. Not implicitly satisfied are criteria C3, C8, C10, and C16-C18, as they need further documentation and discussions to be made, which are not directly related to the technical implementation of a study. As summarised in Table A.6, by following our generic containerised approach and using the Privacy Observatory platform, every reimplemented study improved with at least implicitly satisfying three additional criteria compared to the original analysis depicted in Tables A.2 to A.5.

The specific descriptions of the criteria are given in Table A.1 and the assessment of the original studies under these criteria are shown in the Appendix in Tables A.2 to A.5. In the following section, we discuss 12 satisfied of a total of 18 possible criteria.

5.2.1 Dataset

The following reimplementation criteria were categorized as Dataset criteria by Demir et al. They are concerned with the description and the scope of the targets the study was performed on.

C1: State analysed sites

The first criteria of Demir et al. is satisfied, as the domain-list generator definitions are stored on the Privacy Observatory as `Domainset` objects, and can therefore also be investigated.

C2: State analysed pages

Satisfied, as through the execution and thus the concretisation of a domain-list generator, the `input.txt` can be rendered, which is getting passed to the associated studies and stored.

C4: Perform multiple measurements

Satisfied, as by defining a crontab frequency when creating a new study on the Privacy Observatory, the automatic regular repetition of a study can be automatically registered for execution.

5.2.2 Experiment design (building the crawler)

This category is concerned with concrete specifications about the deployed technology stack of the crawler. As this is precisely specified when writing a Dockerfile and building the Docker image, many of these criteria can be fulfilled implicitly.

C5: Name crawling tech.

Satisfied, since the individually installed components of the crawling engine can be extracted from inspecting the specific layers of the Docker image, as shown in Fig. 5.1. This can be extracted from a Docker image using the `docker history --no-trunc` command. Additionally, if the Dockerfile is also preserved beside the Docker image, the individual build commands are easier to read than extracting Docker image layers.

| | | |
|----|---|-----------|
| 1 | LABEL org.opencontainers.image.ref.name=ubuntu | 0 B |
| 2 | LABEL org.opencontainers.image.version=18.04 | 0 B |
| 3 | ADD file ... in / | 24.5 MB |
| 4 | RUN /bin/sh -c apt-get update | 249.71 MB |
| 5 | RUN /bin/sh -c CHROMEDRIVER_VERSION='curl -sS | 7.22 MB |
| 6 | RUN /bin/sh -c CHROME_SETUP=google-chrome.deb && | 131.15 MB |
| 7 | ENV LANG=C.UTF-8 | 0 B |
| 8 | ENV PYTHONUNBUFFERED=1 | 0 B |
| 9 | COPY ./requirements.txt /opt/anymip/requirements.txt # buildkit | 249 B |
| 10 | RUN /bin/sh -c pip3 install | 25.99 MB |
| 11 | COPY ./ /opt/anymip # buildkit | 35.92 MB |
| 12 | CMD ["/bin/sh" "-c" "cd /opt/anymip | 0 B |

Figure 5.1: Optimized layers of the AnymIP Docker image containing environment variable definitions and system configuration instructions, automatically extracted and visualised by DockerHub.

C6: State adjustments to crawling tech.

Satisfied, similar to C5, the inspection of the Docker image layers reveals all of the performed adjustments to the crawling technology stack.

C7: Describe extensions to crawling tech.

Satisfied, since again with the help of layer inspection of the Docker images, extensions of the crawling technology stack are fully specified.

C9: Used crawler is publicly available

Satisfied, as the default process of deploying a new study on the Privacy Observatory is by pushing the built Docker images to DockerHub and referencing this publicly available Docker image repository in the `docker-compose.yaml` file, which is uploaded to the platform.

5.2.3 Experiment design (experiment env.)

Compared with the previous category, this class of criteria focuses on the accurate description of the study execution environment.

C11: Describe crawling strategy

Satisfied, as it was explained in Section 3.1, the module requirements control and command all crawling executions of the study and thus also define the used crawling strategy.

C12: Document a crawl's location

Satisfied, as the geo-location or VPN configuration of the specific worker, which is linked to a study, can be investigated in the specifications on the Privacy Observatory platform.

C13: State browser adjustments

Satisfied, since the precise configuration and browser profiles can be directly extracted from the layers of the Docker images.

C14: Describe data processing pipeline

Satisfied, as it was explained in Section 3.1, the module requirements include the automation of all post-processing scripts, that need to be performed in order to generate the final measurement outputs of a study. Thus, all post-processing steps are also included in the layers of a Docker image.

5.2.4 Evaluation

Opposed to the above criteria, most criteria in this category are connected to work involving interpretation of the study and can therefore not be automated or directly fulfilled.

C15: Make results openly available

Satisfied, as the raw measurement results of each study run can be viewed on the Privacy Observatory platform and aggregated values can be traced back to the individual measurement of each targeted domain.

Long-term analysis

The studies which we were able to successfully reimplement on the Privacy Observatory platform, were conducted in a regular interval over a period of several weeks. The insights gained during this process are summarised in this chapter, and we also include an analysis of variance over crawls to report the representatives of single-point-in-time measurement.

Since the Privacy Observatory infrastructure in the ETH Cloud was only production-ready in the last few weeks of this thesis, the following measurements were collected using small sample sizes of the first 200 domains from the current Tranco list (auto-pulled by domain-list generators). Also, it was executed on a more lightweight setup of the Privacy Observatory located on a workstation. Thus, these results shall be primarily regarded as proof of concept of the platform. Additionally, in multiple instances, bug hunting was still performed, while collecting repetitive measurements.

6.1 CookieBlock

We were able to reimplement all of the original measurements of the CookieBlock paper, which allows a full comparison of the performance of our modified source code. This study reports eight measurements (with seven of these being privacy violations) and we include also one statistical variable in the following list of observations.

- `perc_cmp_found`: Websites on which a consent management platform is detected.
- `unclassified`: Websites, which do not specify the purpose of some of their cookies.
- `undeclared`: Websites, which do not declare some of their cookies in the cookie banner.

- `incorrect_expiry`: Websites with cookies expiring at least 50.0% longer than declared.
- `ignored_choices`: Websites with cookies set despite negative consent.
- `implicit_consent`: Websites with cookies set prior to user's consent.
- `multiple_purpose`: Websites with cookies labelled for different or contradictory purposes.
- `wrong_purpose_majority`: Websites with misdeclared cookies based on cross-reference between scanner websites.
- `wrong_purpose_wellknown`: Websites with misdeclared cookies based on a library of well-known cookies.

Table 6.1 summarises the aggregated results of the reimplementations from 2023 of the original cookie analysis crawler from 2021 and each number indicates the percentage of websites on which a violation was detected. While most of the results are similar, the `wrong_purpose_majority` as well as the `multiple_purpose` numbers show significant differences compared to the original values. Whereas the `wrong_purpose_majority` can be led back to a mismatching threshold value in relation to the crawled dataset, the discrepancy of the `multiple_purpose` could not be traced back to any technological issues of the reimplementations.

Table 6.1: Measurement results from the original and reimplemented CookieBlock study.

| | 2021 | 2023.07 | 2023.08 | 2023.09 | 2023.10 |
|-------------------------|-------|---------|---------|---------|---------|
| Unclassified | 25.4% | 12.5% | 12.5% | 12.5% | 12.5% |
| Undeclared | 82.5% | 87.5% | 75.0% | 87.5% | 100.0% |
| Incorrect Expiry | 15.5% | 33.3% | 25.0% | 33.3% | 33.3% |
| Ignored Choices | 21.3% | 12.5% | 12.5% | 0.0% | 0.0% |
| Implicit Consent | 69.7% | 80.0% | 87.5% | 100.0% | 100.0% |
| Multiple Purpose | 2.3% | 62.5% | 50.0% | 66.6% | 66.6% |
| Wrong Purpose Majority | 30.9% | 0.0% | 0.0% | 0.0% | 0.0% |
| Wrong Purpose Wellknown | 8.2% | 33.3% | 25.0% | 33.3% | 33.3% |
| Percentage CMP Found | 3.5% | 4.0% | 3.0% | 3.0% | 3.0% |

6.2 OmniCrawl

As part of their PETS 2022 publication, this study released an artefact containing the code of the original crawler on GitHub. However, as result processing was done manually, the delivered post-processing source code only contained Jupyter notebooks. We thus expanded the capabilities of the code for the Brave and Tor browsers.

- `desktop-firefox86-d_firefox86`: Number of third-party advertising and tracking requests performed on page-load on Firefox.
- `desktop-chrome88-d_chrome88`: Number of third-party advertising and tracking requests performed on page-load on Chrome.
- `desktop-brave21-d_brave21`: Number of third-party advertising and tracking requests performed on page-load on Brave.
- `desktop-tor-d_tor`: Number of third-party advertising and tracking requests performed on page-load on Tor.

The measurements originating from the reimplemented and containerised OmniCrawl study are shown in Table 6.2, which are given as absolute numbers in contrast to the measurement results of the previous study. The number of recorded third-party advertising and tracking requests on Firefox and Chrome stayed in the same range as in the original paper. A significantly higher number of these requests were measured for the Tor browser, and while no data relating to the desktop version of Tor were provided, the ones for the mobile setup were included in the OmniCrawl paper. Additionally, the paper traces this high number of requests back to the design of Tor. Tor was developed with the goal of users having the exact same fingerprint. As a result, the Tor browser does not include any advertisement or tracking-blocking features.

Table 6.2: Comparison of the original crawl with the reproduced and extended study crawler of OmniCrawl.

| | 2021 | 2023.08 | 2023.09 | 2023.10 |
|---------|---------|---------|---------|---------|
| Firefox | 20.0 | 31.4 | 20.4 | 17.8 |
| Chrome | 15.0 | 12.2 | 12.4 | 14.9 |
| Brave | No data | 17.2 | 13.3 | 17.4 |
| Tor | No data | 31.4 | 20.4 | 37.0 |

6.3 AnymIP

Since the entire code basis was rewritten for the reimplementation as we were not able to get hold of the original source code, we extracted measurements from the paper that we set as a target for recrawling.

- `ga_detected`: Google Analytics is detected in the HTTP requests of the domain
- `no_ip_anonymization`: The detected Google Analytics requests do not use IP anonymization

In comparison to the reimplemented studies above, only one crawling measurement is contained in the original paper, which is the percentage of websites on which this specific Google Analytics misconfiguration was identified. Whilst 13% of the crawled websites use Google Analytics, 7% of all analysed websites contain a privacy violation regarding the non-anonymisation of IP addresses of website visitors as shown in Table 6.3. As this number is substantially lower than the value from 2019, this can be led back to only crawling the first 200 domains of the Tranco list, which often indicates a higher degree of compliance with legal requirements.

Table 6.3: Reimplemented AnymIP Study results of repetitive measurements.

| | 2019 | 2023.07 | 2023.08 | 2023.09 | 2023.10 |
|---------------------|---------|---------|---------|---------|---------|
| GA Detected | No data | 13.0% | 17.5% | 16.0% | 13.0% |
| No IP Anonymization | 12.7% | 5.7% | 7.0% | 6.0% | 7.0% |

6.4 LeakyForms

As mentioned in Section 5.1.7, we needed to rewrite the leak detection post-processing module in order to gain results from the crawler after fixing multiple errors in the source code. Thus, the results were expected to be significantly different from the original measurements.

Whilst the post-processing scripts of this study generate four different types of leaks besides `url_leaks`, and `post_leaks`, the LeakyForms paper only provides us with one original measurement value, which is the percentage of websites, which contained any types of leaks.

- `generic_leak`: Websites that contain any type of leak that we were able to detect.

Table 6.4 shows a significant deviation from the 2023 `generic_leaks` measurement value regarding the original value of 2021. Besides being related to the fact that only popular domains were scanned, which do not use dubious practices such as leaks in the forms to gather additional information, the experienced deviation can also be caused by our replacement of the original detection algorithm. Even if we made efforts to identify Base64 leak detections as well as HTML URL Encoding, there remains a possibility of leaks in other unsupported formats.

Table 6.4: Original and reimplemented LeakyForms measurement results.

| | 2021 | 2023 |
|---------------|------|------|
| Generic Leaks | 3.0% | 1.5% |

Discussion

In this chapter, we provide guidance for future application of our Privacy Observatory platform. We also discuss the limitations of the platform and the ethical implications of our work.

7.1 Ethics

We identify three types of risks stemming from empirical privacy research of websites: risk of incurring costs on website operators, legal risks stemming from trespassing regulations, and reputation harm caused to websites. We discuss these risks in detail below.

Most privacy-focused web crawlers, especially the ones we reimplemented in this thesis, only create negligible traffic on visited websites, which was also noted by Demir et al. [4]. The major financial impact is caused by seeing ads that drain the budget of companies which are paying for advertisements through providers like Google Ads. Since every domain is only visited a concrete number of times per study run, we accept these minor issues.

Concerning the legal perspective, we assessed various legal frameworks and determined that our research complies with laws pertaining to fraud, trespass, or breach of contract.

Finally, the most considerable risk stems from reputation harm. Given that we are not authors of the reproduced studies, we cannot guarantee that their results are not falsely accusing websites of privacy violations. This risk must therefore be thoroughly assessed, ideally by an institutional ethics review, prior to publicly disclosing our results at a per-site level. We therefore release only aggregated results in this thesis, and the Privacy Observatory platform remains accessible only to authenticated users we limit to people directly involved in the project.

7.2 Limitations

We identified several limitations of the platform that could not be corrected due to technical, conceptual, or simple time constraints. They may however be improved or fully eliminated in future releases of the platform.

7.2.1 Long-term stateful crawls

Due to the approach of using Docker images for storing the base configuration, all modifications to the browser profiles, including its history and cookies, are disregarded after the completion of a full study run. Thus, long-term stateful crawls, which for example want to observe the influence of returning to the same website after several months, are not feasible using the Privacy Observatory and would require a storage-intensive Docker container export and archiving or a shared network storage mount on all workers.

7.2.2 Host per worker

In order to make the deployment of study workers as easy as possible, it was decided to design those as Docker images as well, which can be launched through a standardised `docker-compose.yaml` file with the Docker socket mounted inside the container for Docker control, as explained in Section 4.2.2. Whilst this design would, in theory, allow multiple Docker socket operators (workers) on the same node, as these do not interfere with each other, isolated data interfaces for input and output turned out to be more difficult than anticipated, since the workers are Docker containers itself and cannot simultaneously mount `input.txt` and `output.txt` files. Like this, the only option was to define an absolute path in the root file system which gets mapped inside of the worker container as well as in the containers of the executed studies.

Since this absolute path is hard-coded inside the `docker-compose.yaml` files of the individual studies and no dynamic variable system have yet been introduced to the Privacy Observatory, there can not be two worker containers running on the same host, since they would overwrite their respective input and output files.

In the future, this limitation could be tackled, as explained above, by introducing placeholders in the `docker-compose.yaml` files, which are stored inside the study objects on the platform, and get replaced by the individual worker IDs. Then again, the workers would need to be mapped to the individual paths of their respective data interface files, which raises the complexity when deploying workers on new hosts.

7.2.3 Distributed / parallel studies

Due to the fact, that in the process of executing a study, a `docker-compose.yaml` file is taken and given to a worker as a command sequence, it is not possible, to create a setup with multiple Docker workers working distributed on the same study with the current concept. The same applies to the Docker built-in scaling feature, which can only be achieved by defining a service multiple times in the `docker-compose.yaml` file at the time of writing this thesis.

This limitation could potentially be solved by letting a user define multiple `docker-compose.yaml` files per study and modifying the queuing system, such that it respects the dependencies within an individual study related to a multiple of these command files. This not only raises the level of complexity of the job scheduler but also introduces new issues regarding worker timings (i.e. one worker would need to serve a database as long as the other workers are actively using the database).

7.3 Future work

Besides the above-mentioned mitigations of the identified limitations of the Privacy Observatory, we propose the following improvements in future work.

7.3.1 Mobile platforms

Cassel et al. [15] highlighted in the OmniCrawl paper that different measurements performed on desktop systems as well as on mobile devices often exhibit large discrepancies.

Even though implementing mobile-based studies in a containerised Android container was outside of the scope of this thesis, proof of concept was performed with Docker images based on Android.¹ Besides the typical challenges highlighted in Chapter 5, the same approaches could be used by having a Chrome WebDriver² to navigate websites and perform measurements. On the downside, the full communication would need to be routed through the Android Debug Bridge (*adb*), which has been proved unstable for long-term running observations in the past.

¹<https://github.com/budtmo/docker-android>

²<https://chromedriver.chromium.org/getting-started/getting-started—android>

7.3.2 (Very) Long-term analysis

Even though the studies on the Privacy Observatory only carried out repetitive crawls and measurements for a few months, we are already able to recognise clear trends with the baseline of the measured values in the original papers. Although we tried to keep it as similar as possible, the containerisation itself changed the execution environment.

Even after this thesis, the Privacy Observatory can continue analysing the measurement results from the studies over several years. The freeze effect of the Docker images and the guaranteed constant execution environments would enable the recognition of trends even more precisely, especially in combination with automatically generated ranked domain-lists using the domain-list generators. An analysis of the platform's long-term measurement results could therefore reveal new information about the privacy status on the internet. It can measure the influence of new regulations, significant fines issued for non-compliance, or new privacy technologies.

Conclusion

In this master thesis report, we introduced a practical approach of enabling long-term reproducibility of internet privacy studies as well as implemented the Privacy Observatory platform as an orchestrator, to facilitate long-term regular crawls and measurements.

We selected five influential studies to reimplement using our proposed containerised approach through the use of Docker images, a standardised input and output interface, and a central controller to perform the web measurements and post-processing steps fully autonomously. Demir et al. [4] defined a total of 18 reimplementation criteria, which form the theoretical requirements of web measurement study reproducibility. We evaluate the compliance of our five selected studies with these criteria and highlight the implicit fulfilment of these criteria by following our containerisation method.

Additionally, we state a total of six practical principles, based on discovered reappearing issues while reimplementing, that shall be closely followed in future studies to facilitate long-term reproducibility of the studies at a later point in time by another research group.

After having built these reimplemented studies as Docker images, we registered them on the Privacy Observatory and performed regular study runs. This enables us to observe long-term trends regarding compliance with privacy regulations in a constant and immutable execution environment.

Bibliography

- [1] Ramal Moonesinghe, Muin J Khoury, and A Cecile J W Janssens. “Most published research findings are false—but a little replication goes a long way”. In: *PLoS medicine* 4.2 (2007), e28.
- [2] Andy Cockburn et al. “Threats of a replication crisis in empirical computer science”. In: *Communications of the ACM* 63.8 (2020), pp. 70–79.
- [3] Nurullah Demir et al. “On the Similarity of Web Measurements Under Different Experimental Setups”. In: *Proceedings of the 23rd ACM Internet Measurement Conference, 23rd ACM Internet Measurement Conference (2023) Montreal, Kanada, 24.10. 2023–26.10. 2023*. 2023.
- [4] Nurullah Demir et al. “Reproducibility and replicability of web measurement studies”. In: *Proceedings of the ACM Web Conference 2022*. 2022, pp. 533–544.
- [5] Terry Benzel. “Security and Privacy Research Artifacts: Are We Making Progress?” In: *IEEE Security & Privacy* 21.01 (2023), pp. 4–6.
- [6] Franziska Roesner, Tadayoshi Kohno, and David Wetherall. “Detecting and Defending Against {Third-Party} Tracking on the Web”. In: *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. 2012, pp. 155–168.
- [7] Steven Englehardt and Arvind Narayanan. “Online tracking: A 1-million-site measurement and analysis”. In: *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 2016, pp. 1388–1401.
- [8] Gunes Acar et al. “The web never forgets: Persistent tracking mechanisms in the wild”. In: *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*. 2014, pp. 674–689.

-
- [9] Martino Trevisan et al. “4 years of EU cookie law: Results and lessons learned”. In: *Proceedings on Privacy Enhancing Technologies* 2019.2 (2019), pp. 126–145.
- [10] Ryan Amos et al. “Privacy policies over time: Curation and analysis of a million-document dataset”. In: *Proceedings of the Web Conference 2021*. 2021, pp. 2165–2176.
- [11] Maximilian Hils, Daniel W Woods, and Rainer Böhme. “Measuring the emergence of consent management on the web”. In: *Proceedings of the ACM Internet Measurement Conference*. 2020, pp. 317–332.
- [12] Adrian Dabrowski et al. “Measuring cookies and web privacy in a post-gdpr world”. In: *Passive and Active Measurement: 20th International Conference, PAM 2019, Puerto Varas, Chile, March 27–29, 2019, Proceedings 20*. Springer. 2019, pp. 258–270.
- [13] Dino Bollinger et al. “Automating Cookie Consent and {GDPR} Violation Detection”. In: *31st USENIX Security Symposium (USENIX Security 22)*. 2022, pp. 2893–2910.
- [14] Kostas Drakonakis, Sotiris Ioannidis, and Jason Polakis. “The cookie hunter: Automated black-box auditing for web authentication and authorization flaws”. In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2020, pp. 1953–1970.
- [15] Darion Cassel et al. “Omicrawl: Comprehensive measurement of web tracking with real desktop and mobile browsers”. In: *Proceedings on Privacy Enhancing Technologies* 2022.1 (2021).
- [16] Max Maass et al. “Effective notification campaigns on the web: A matter of trust, framing, and support”. In: *30th USENIX Security Symposium (USENIX Security 21)*. 2021, pp. 2489–2506.
- [17] Asuman Senol et al. “Leaky Forms: a study of email and password ex-filtration before form submission”. In: *31st USENIX Security Symposium (USENIX Security 22)*. 2022, pp. 1813–1830.
- [18] Victor Le Pochat et al. “Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation”. In: *Proceedings of the 26th Annual Network and Distributed System Security Symposium*. NDSS 2019. Feb. 2019.
- [19] Tobias Urban et al. “Measuring the Impact of the GDPR on Data Sharing”. In: *Proceedings of the 15th ACM Symposium on Information, Computer and Communications Security (AsiaCCS’20)*. ACM Press, New York, NY, USA. 2020.
- [20] Tobias Urban et al. “Beyond the front page: Measuring third party dynamics in the field”. In: *Proceedings of The Web Conference 2020*. 2020, pp. 1275–1286.

- [21] Zhiju Yang and Chuan Yue. "A comparative measurement study of web tracking on mobile and desktop environments". In: *Proceedings on Privacy Enhancing Technologies* 2020.2 (2020).

Appendix

A.1 Reimplementation criteria

The Table A.1 contains the descriptions of the reimplementation criteria stated by Demir et al. [4], whilst assessments of the original papers can be found in Tables A.2 to A.5. Additionally, the criteria, which are implicitly satisfied by following our generic containerised approach and using the Privacy Observatory platform are summarised in Table A.6.

A.1. Reimplementation criteria

Table A.1: Description of the reimplementation criteria according to Demir et al. [4].

| Category | Criterion ID | Description |
|--|--------------|---|
| Dataset | C1 | State analysed sites: States used dataset, toplist, or user clickstreams, including version. |
| | C2 | State analysed pages: Offers a .csv or comparable with all analysed pages (i.e. distinct URLs). |
| | C3 | State site or page selection: Discusses the selection process of analysed sites. |
| | C4 | Perform multiple measurements: Discuss which pages are analysed in consecutive measurement runs, if appropriate. |
| Experiment Design (Building the Crawler) | C5 | Name crawling tech.: Describes the used crawling technology (e.g. OpenWPM). |
| | C6 | State adjustments to crawling tech.: States which technology features were used and/or (slightly) adjusted. |
| | C7 | Describe extensions to crawling tech.: Describes which features were developed to conduct, if any. |
| | C8 | State bot detection evasion approach: Discusses which means were taken that the crawler was not detected, if necessary. |
| | C9 | Used crawler is publicly available: Provides the crawler in a public location. |
| | C10 | Mimic user interaction: Describes how the user interaction was implemented, if applicable. |
| Experiment Design (Experiment Env.) | C11 | Describe crawling strategy: Describes which crawling strategy was used (e.g. stateless vs. stateful). |
| | C12 | Document a crawl's location: States from which location(s) the study was conducted. |
| | C13 | State browser adjustments: Discusses properties of the browser (e.g. user agent, version, used extensions). |
| | C14 | Describe data processing pipeline: Describes the data processing steps in detail. |
| Evaluation | C15 | Make results openly available: Authors provide the (raw) measurement results. |
| | C16 | Provide a result/success overview: Describes the outcome of the measurement process on a higher level. |
| | C17 | Limitations: Discusses the limitations of the experiment. |
| | C18 | Ethical discussion: Discusses ethical implications of the experiment (e.g. exploiting vulnerabilities). |

A.1. Reimplementation criteria

Table A.2: Reimplementation criteria assessment of the CookieBlock paper.

| Criterion ID | CookieBlock |
|--------------|--|
| C1 | ✓ Tranco ranking [32] of May 5th, 2021 |
| C2 | ✓ https://tranco-list.eu/list/P63J/full |
| C3 | ✗ |
| C4 | ✓ For each domain, after arriving on the landing page, the crawler detects which CMP is actively present on the site. Then, the set of declared cookies is extracted. If this proceeds without error, the subsequent steps are intended to trigger the creation of cookies in the browser. First, the crawler consents to all cookie purposes in the cookie banner using the Consent-O-Matic extension [27, 39]. This is required, as otherwise, the lack of consent would prevent cookies from being created. Afterwards, the browser visits random links leading to subpages of the domain, scrolling down to the bottom of each page and performing random cursor movements for each subpage. Urban et al. [51] reported that browsing subpages increases the number of observed cookies up to 36%. As a trade-off between crawling speed and the amount of collected data, we visit ten randomly selected subpages for each site. |
| C5 | ✓ OpenWPM |
| C6 | ✗ |
| C7 | ✓ Extended to handle data extraction from the CMPs. |
| C8 | ✓ The probability with which advertising cookies will evade detection can be identified using the recall metric of the advertising class. The potential to break essential functionality on websites can be found in the recall of the necessary category. The closer either performance metric is to 1 or the lower the privacy threat, respectively, the less likely a website is to break. |
| C9 | ✓ https://karelkubicek.github.io/post/cookieblock |
| C10 | ✓ Afterwards, the browser visits random links leading to subpages of the domain, scrolling down to the bottom of each page and performing random cursor movements for each subpage. |
| C11 | ✗ |

A.1. Reimplementation criteria

| Criterion ID | CookieBlock |
|--------------|---|
| C12 | <p>✓</p> <p>Our scan was performed on an AWS EC2 server instance located in Germany, with 32 vCPUs, 64 GB of RAM, and a 10 Gigabit connection. Special care was taken to perform the scan from within an EU country, as previous works have shown that there is significant geographic discrimination with regards to GDPR enforcement.</p> |
| C13 | <p>✓</p> <p>Note that while CookieBlock imitates the behavior of a CMP, it is not intended to interact with or remove the cookie banners shown on websites. This function is already fulfilled by existing browser extensions, such as Consent-O-Matic [27], which can be used in conjunction with CookieBlock. CookieBlock also does not act as a replacement for the cookie banner in the legal sense, and its use is not a justification for websites to skip the gathering of user consent.</p> |
| C14 | <p>✓</p> <p>Observed violations</p> |
| C15 | <p>✗</p> |
| C16 | <p>✓</p> <p>6 Observed violations</p> |
| C17 | <p>✓</p> <p>7 Limitations</p> |
| C18 | <p>✗</p> |

A.1. Reimplementation criteria

Table A.3: Reimplementation criteria assessment of the CookieHunter paper.

| Criterion ID | CookieHunter |
|--------------|---|
| C1 | ✓ Alexa Top 1 million (09/14/2017) |
| C2 | Undoc. (Top-1K hand-picked / Top-1K randomly selected / Any-rank randomly selected) |
| C3 | ✗ |
| C4 | ✓ The crawl starts at the landing page and goes to a depth of 2 – we opt for a more shallow crawl to reduce the crawl’s duration and enable our large-scale study. Our framework collects all links included in each page that point to the same domain, and subsequently visits and inspects them. This step prioritizes links that contain an account-related keyword (e.g. signin, register etc.) and follows a breadth-first search (BFS) approach. If both types of forms are yet to be found, the final step is to collect the first 30 links from the homepage and inspect them, excluding previously-visited URLs. This is based on the intuition that such pages are typically easily accessible to users and not hidden behind multiple menus, and are usually at the top of the page. |
| C5 | ✓ Selenium + Chrome and Firefox WebDrivers |
| C6 | ✗ |
| C7 | ✓ Develop XDriver, a custom browser automation tool designed for security-oriented tasks that offers improved fault-tolerance during prolonged black-box interactions with web apps. XDriver is built on top of Selenium and the official Chrome and Firefox WebDrivers. |
| C8 | ✗ |
| C9 | ✗ |

A.1. Reimplementation criteria

| Criterion ID | CookieHunter |
|--------------|---|
| C10 | <p>✓</p> <p>For each URL, we iterate over the candidate SSO elements and click them. We prioritize elements that are displayed, based on the intuition that sites are usually upfront about the available login options. For displayed elements we use Selenium’s click method, effectively replicating a user’s action. For hidden elements we refrain from trying to make those elements appear, which would involve clicking over other elements and potentially leading to unintended behavior and considerably increasing the process’ duration. Instead, we try to trigger their onClick method via JavaScript. While this is generally effective, in some cases the candidate element is an outer wrapper element (e.g. a div-tag which contains an a-tag), and clicking it via JavaScript will not trigger SSO.</p> |
| C11 | <p>✓</p> <p>Specifically, it launches a new browser instance, reloads the current browser profile to maintain state and updates its own object reference with that of the new one, so as to transparently update all references of the driver held by the framework modules.</p> |
| C12 | ✗ |
| C13 | ✗ |
| C14 | <p>✓</p> <p>EXPERIMENTAL EVALUATION</p> |
| C15 | ✗ |
| C16 | <p>✓</p> <p>4 EXPERIMENTAL EVALUATION</p> |
| C17 | ✗ |
| C18 | <p>✓</p> <p>Captchas ethical discussion</p> |

A.1. Reimplementation criteria

Table A.4: Reimplementation criteria assessment of the OmniCrawl paper.

| Criterion ID | OmniCrawl |
|--------------|---|
| C1 | ✓ Tranco website ranking |
| C2 | ✓ https://tranco-list.eu/list/4ZWX |
| C3 | ✗ |
| C4 | ✗ |
| C5 | ✓ Mitmproxy |
| C6 | ✗ |
| C7 | ✓ Inject scripts to instrument JavaScript APIs. |
| C8 | ✓ We use custom scripts instead of Selenium [15], a web automation tool widely used by previous work [8, 33, 43], because websites may be able to detect the use of Selenium via the existence of modified JavaScript properties [88]. |
| C9 | ✓ https://github.com/OmniCrawl |
| C10 | ✓ We include one version of Chrome that scrolls down the page in the last 45 seconds of a site visit. |
| C11 | ✓ A browser profile is a set of cookies and local storage representing a user’s client-side state. Like prior work [43], we built a seed profile for each browser and reset the browser state to it before each website visit so that the order of website visits does not affect the data we collect. |
| C12 | ✓ We collected data from two different locations: the United States, in North America (NA), and Taiwan, in Asia (AS). |
| C13 | ✓ However, the emulation in that work was limited to changing desktop Firefox to use the User-Agent string and screen resolution of mobile Firefox. Other modifications were not implemented, e.g. sensor API results were not spoofed, although they are often used to fingerprint mobile devices. |
| C14 | ✓ Results |
| C15 | ✗ |
| C16 | ✓ 4 Results |

A.1. Reimplementation criteria

| Criterion ID | OmniCrawl |
|---------------------|------------------|
| C17 | X |
| C18 | X |

A.1. Reimplementation criteria

Table A.5: Reimplementation criteria assessment of the AnymIP paper.

| Criterion ID | AnymIP |
|--------------|---|
| C1 | ✗ German Wikipedia, filtered on the Top-Level Domain (TLD) .de |
| C2 | ✗ |
| C3 | ✗ |
| C4 | ✗ |
| C5 | ✓ Chromium browser + DevTools protocol |
| C6 | ✓ We modify the user agent to hide that Chromium was running headless. |
| C7 | ✗ |
| C8 | ✗ |
| C9 | ✗ |
| C10 | ✓ We also scroll the page for a random amount in short random intervals, since additional GA requests might be sent on site usage. |
| C11 | ✗ |
| C12 | ✗ |
| C13 | ✗ |
| C14 | ✓ Methodology |
| C15 | ✗ |
| C16 | ✓ 5 Results |
| C17 | ✓ 7.4 Limitations |
| C18 | ✓ 4.8 Ethical Considerations |

A.1. Reimplementation criteria

Table A.6: Implicitly satisfied reimplementation criteria by following a containerised approach and using the Privacy Observatory platform.

| Criterion ID | Generic Privacy Observatory Study |
|--------------|--|
| C1 | ✓ Domain-list Generator definition |
| C2 | ✓ Domainlist Generator concretisation |
| C3 | Not implicitly sat. |
| C4 | ✓ Multiple repetitions of a study are automatically performed, based on the selected crontab frequency. |
| C5 | ✓ Installed crawling components can be extracted from the Docker image layers. |
| C6 | ✓ Customised configuration steps can be extracted from Docker image layers. |
| C7 | ✓ Additional extensions can be extracted from the Docker image layers. |
| C8 | Not implicitly sat. |
| C9 | ✓ If DockerHub is used (as is the default), the full Docker images and thus also the crawler is publicly available. |
| C10 | Not implicitly sat. |
| C11 | ✓ By using Docker images, the full crawler is automatically reset after each study run. |
| C12 | ✓ The location or VPN configuration of the linked worker. |
| C13 | ✓ Configurations of crawling components can be extracted from the Docker image layers. |
| C14 | ✓ The post-processing steps are directly included in the study containers. |
| C15 | ✓ The raw results of each study run can be investigated down to the individual domains. |
| C16 | Not implicitly sat. |
| C17 | Not implicitly sat. |
| C18 | Not implicitly sat. |

A.2 Code base artifact

We decided to publish the code bases of this thesis as artefacts on GitHub as well as the individual images of the reimplemented studies on DockerHub. In the following, we provide the description of each published artefact as well as the link to the repositories.

A.2.1 Privacy Observatory - GitHub repository

The source code of the components of the Privacy Observatory can be found on the following GitHub repository in different subfolders.

- <https://github.com/PatriceKast/privacy-observatory>

Each of the components is also fully containerised and can be started either by using the individual `docker-compose.yaml.template` files, contained in the specific folders, or by using the global `docker-compose.yaml.template` file, included in the root directory of the repository. These template files need to be copied and the correct environment variables need to be set.

A.2.2 Reimplemented studies - DockerHub image repository

The successfully reimplemented studies can be found on the individual DockerHub repositories, which are publicly available:

- <https://hub.docker.com/repository/docker/kastpatrice/cookieblock>
- <https://hub.docker.com/repository/docker/kastpatrice/omnicrawl>
- <https://hub.docker.com/repository/docker/kastpatrice/anymip>
- <https://hub.docker.com/repository/docker/kastpatrice/leakyforms>

All of these Docker images follow the generic interface requirements and principles defined in Section 3.1. In order to run them, the `docker-compose.yaml` file in Listing A.1 can be used to handle input and output and it can be started using the command `docker-compose up --force-recreate`.

```
1 version: "3.4"
2 services:
3   leakyforms:
4     image: leakyforms:latest
5     build:
6       dockerfile: Dockerfile
7       context: ./
8     volumes:
9       - /opt/input.txt:/opt/leakyforms/input.txt
10      - /opt/output.txt:/opt/leakyforms/output.txt
```

Listing A.1: docker-compose.yaml file for running a containerised study following the generic interface definition