MASTER'S THESIS

Privacy Patterns in Practice

Vek, J.

Award date:
2022

PRIVACY PATTERNS IN PRACTICE

by

Jasper Vek

Student number:	852029043	
Thesis committee:	dr. Greg Alpár (supervisor and chairman)	Open University
	dr. Ebrahim Rahimi (2nd supervisor)	Open University

CONTENTS

1. GLOSSARY

Design pattern	a recognized reusable solution to a specific privacy problem
Privacy design pattern	a design pattern with regards to privacy
PET	Privacy enhancing technology: a concrte technical solution to a known privacy problem
Cryptographic hash-function	function which maps data to a hash value
SHA-256	A cryptographic hash function to generate a HMAC
HMAC	Hash-based Message Authentication Code
Clear text	raw text in an unencrypted form
Cipher text	raw text that has been encrypted
IV	An Initializion vector that is used for data encryption with a secret key

2. Abstract

Privacy design patterns are suggested solutions to possible privacy problems. These could possibly be used to assess possible privacy risks in a software system, by determining which patterns were possible and which have been implemented. In this study a first (manual) method was set up to do so, using the documentation and source code to study the data and overall systems at a high-level. Here, the user-data Synchronization function of Firefox and Chromium were studied to identify possible and implemented privacy design patterns. From a small list of privacy design patterns that were investigated, four were possible in both systems and one has been implemented, without the former patterns posing a significant risk to privacy. In conclusion, this study tried to explore a way to study privacy design patterns in practice.

3. Introduction

Since the introduction of the General Data Protection Regulation (GDPR) [1] in 2018 more emphasis has been placed on privacy with regards to software systems. This also increased interest in research surrounding privacy and the practical applications surrounding privacy within software systems. One way to increase the adherence to the GDPR is by applying Privacy by Design (PbD).

The objective of Privacy by Design is to embed privacy into a software system through the system's life cycle. It proposes seven high-level principles like 'Proactive not reactive; Preventative not reactive'. Privacy by Design is however too vague Van Audenhove et al. [2011] and not practical enough for concrete guidelines regarding software systems because it may not be detailed enough to steer this software development process in a concrete way.

There are more detailed (and practical) concepts that can be linked to Privacy by Design like privacy strategies, privacy tactics, privacy design patterns and privacy enhancing technologies (PETs). These range from concepts like minimizing and hiding data to increase its privacy sensitivity at the privacy strategy level, to concrete implementations of privacy enhancing technologies like implementing an onion network node infrastructure. These kind of concepts and implementations open the door to being able to investigate a system with regards to privacy because the problems and solutions to privacy problems get more clear the more detailed the investigation goes. In this study privacy design patterns were chosen to be used as a way to investigate software systems with regards to privacy.

Privacy design patterns are concrete solutions to privacy problems. They can be compared to standard software design patterns and security design patterns. These patterns can be found on privacypatterns.org and include a summary, context, problem, solution and example section to describe a specific privacy design patterns. An example is the Location Granularity pattern [2] that discusses the problem surrounding location data and its specificity. The solution and examples are focused on how to make the location data more abstract, as to limit privacy risks in these kind of scenario's in software systems. In this way a privacy design pattern ties together a recognizable scenario with regards to privacy and proposes a solution that is backed up by academic sources, while also recognizing more general concepts like abstracting data (privacy strategy). This makes privacy design pat-

[1] https://eur-lex.europa.eu/eli/reg/2016/679/oj
[2] https://privacypatterns.org/patterns/Location-granularity

terns interesting in the study of software systems regarding privacy and determining problems, risks and possible solutions to privacy related scenario's.

To study a software system in this way was also the goal of this study: to recognize privacy design patterns in existing software systems and to explore the privacy problems surrounding these privacy design patterns. Determining the contexts of the privacy design patterns and its accompanying problems can say something about the risks in a specific software system; recognizing the implementation of the privacy design patterns that can solve these problems in a system can say something about privacy in a specific system.

The systems that were studied were the user-data synchronization methods of Firefox and Chromium. These are two popular open source web browsers that both implement a relevant functionality surrounding privacy with its synchronization function. Studying the source code itself was important to back up claims and documentation surrounding the synchronization functionality.

A method was set up in the VAF phase to study a client-server application like Firefox and Chromium Sync, which was used in this study. An important sub-part of the study was also to explore this method and how practical this endeavour of studying a software system with regards to privacy design pattern is, and how this method can be applied in more scenario's than a purely client-server setting.

4. Research

4.1. Research Questions

The goal of the research was to use privacy design patterns to study software systems and explore the privacy risks and solutions in this context. The Firefox and Chromium user-data synchronization functionality was used to explore this goal, where the documentation and source code was studied to recognize the privacy design patterns in question. The privacy design patterns studied were a subset of the ones that are available on privacypatterns.org, concerning the categories Minimize, Abstract, Separate and Hide (see Appendix). Another goal was to explore the use of this kind of approach and to test the method that was set up in earlier work to achieve this goal.

The research questions were as follows:

RQ1: Which data-oriented privacy design patterns are relevant in the context of the data-synchronization function of Firefox and Chromium.

RQ2: What are the privacy threats in Firefox and Chromium related to the context-relevant privacy design patterns?

RQ3: Which data-oriented privacy design patterns are present in the data-synchronization function of Firefox and Chromium, and what does this say about the protection of the data in these functions?

RQ4: How can the proposed method be generalized to finding the applied privacy design patterns in other applications?

The first three research questions are concerned with the study of Firefox and Chromium synchronization, while the fourth is concerned with the method that was used to study these systems.

The results of the study are to test the method and determine its usefulness, including the use of privacy design patterns as a way to explore privacy or recognize privacy threats in a system; to recognize both the relevant and the implemented privacy design patterns

in Firefox and Chromium Sync, including its risks; comparing Firefox and Chromium sync with the privacy design patterns in mind; generalizing the method for future use.

5. Background

Because of the goal of this study, several concepts were explored for the study of a software system and determining privacy risks/threats. In the software engineering field of design patterns and the security field of security design patterns, parties are trying to find these patterns in software as to identify common solutions to software problems. In privacy this is not as common, and concepts had to be found in the VAF phase to study a software system. In short, concepts like privacy, privacy by design, privacy strategies and tactics are too general to study a software system in the scope of this study, and privacy design patterns were chosen because of its more concrete nature, though still abstract. The other concepts will be explained here as background information.

5.1. Privacy

Privacy is a context specific definition that can be described in general as the right to be let alone while also having control over personal information Solove [2002]. The GDPR states in article 4 that: ""personal data' means any information relating to an identified or identifiable natural person ('data subject'); an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an on-line identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person;" [3]

Information is of course needed in the proper use of software systems; this could mean that personal information from a user is shared directly or indirectly with a software system, which then has to be protected if the user would deem this as privacy sensitive information. Some guidelines as of how to handle privacy sensitive data in a good way are the GDPR and ISO 29100 [4]. There is however a gap between these high-level regulations and concrete practices in software systems. The GDPR points to privacy by design as a more concrete way to implement privacy friendly practices.

5.2. Privacy by Design

"Privacy by design (PbD) is an approach whose objective is to discover, represent, implement and manage the rules and tasks that preserve the data privacy of any stakeholder of a software system. PbD should be considered from the project inception phase and throughout the entire software life cycle Morales-Trujillo et al. [2018]. It is comprised of seven steps:

- Proactive not reactive; Preventative not reactive
- Privacy as the Default
- Privacy Embedded into Design
- Full functionality-Positive Sum, Not Zero-Sum
- End-to-End Life cycle Protection
- Visibility and Transparency

[3] https://eur-lex.europa.eu/eli/reg/2016/679/oj
[4] https://www.iso.org/standard/73722.html

- Respect for User Privacy Cavoukian et al. [2009] By itself these concepts are too vague Van Audenhove et al. [2011] to implement privacy in a software system.

5.3. Privacy strategies

Privacy strategies and tactics are high-level concepts that are more detailed than privacy by design and try to make privacy by design more concrete for implementation in software systems Hoepman [2014]. Privacy strategies were derived from the ISO 29100 and Organisation of Economic Co-Operation and Development (OECD) for Economic Co-operation and Development [1980]. There are eight privacy design patterns: Minimize, Separate, Abstract, Hide, Inform, Control, Enforce and Demonstrate. Each privacy strategy outline overall guiding principles for software engineering with regards to privacy. Hide, for instance, proposes to obfuscate data in a way that makes the usefulness of gaining information out of data for a possible attacker harder or impossible.

5.4. Privacy tactics

Privacy tactics are more concrete instances to the privacy strategies Colesky et al. [2016]. These divide a privacy strategy in steps that are even more concrete. The Minimize strategy has the following tactics: Select, Exclude, Strip, Destroy. The Select tactic states: "Select only relevant people and relevant attributes. Determine beforehand which people and which attributes are relevant, and process only that data. These concepts are more concrete than privacy by design and the privacy strategies but are still fairly hard to detect in a system in a concrete way as to determine privacy risks or faults.

5.5. PETs

“Privacy-Enhancing Technologies is a system of ICT measures protecting informational privacy by eliminating or minimizing personal data thereby preventing unnecessary or unwanted processing of personal data, without the loss of the functionality of the information system.” Danezis et al. [2015] PETs are very concrete solutions to privacy design problems which implement an implicit or explicit privacy design pattern. Because of their nature, PETs relate to the implementation phase of software engineering. PETs have however not been implemented much outside of academia as a standard in system design Danezis et al. [2015] and haven’t found their way into system engineers’ toolboxes Gürses and del Alamo [2016]. Because of this, the chance of finding PETs in a system and finding privacy threats in a small use case like Firefox and Chromium Sync is also small.

5.6. Privacy design patterns

Privacy design patterns are inspired by software engineering design patterns Gamma [1995]. Privacy design patterns are actionable expressions of privacy principles but don’t insist on particular solutions or practices Doty and Gupta [2013]. Privacy design patterns are more abstract than software engineering design patterns but still propose a general solution to a possible problem for privacy. These can be found at privacypatterns.org.

A privacy design pattern is at this stage comprised of a description: summary, context, problem, solution, examples. There are no models for these patterns as they are not that concrete as of yet. They are however concrete enough to being able to find them in a software system because they are comprised of a clear enough description, especially the

problem and solution parts of the pattern. They are also broad enough to be applicable for studying every type of software system that could be considered as handling privacy intensive data that could benefit from privacy friendly best practices.

5.7. Approaches to finding privacy design patterns in a system

There is literature surrounding finding software engineering design patterns in code in an automated way; the same goes for security design patterns. For privacy design patterns however, there doesn't seem to be an approach that is catered to privacy design patterns and finding them in a software system.

In the security patterns literature there is an approach for finding security design patterns in code by combining reverse engineering and a manual approach at architecture level Berger et al. [2011]. The automated part is turning the source code into a high-level model, which then can be matched to the model of the pattern that is searched for in the code. The automated modeled information of the system is manually corrected as to better represent reality. A likewise approach was not feasible for this study because of its scope, which is why a manual approach was chosen.

STRIDE is a security threat modeling method that was coined by Microsoft [5]. This is used to analyze a system with regards to security risks and uses modeling and a standard threat catalog to analyze a software system. The overall goal is to determine risks and propose mitigations which can be implemented with regards to security. In the modeling part a Data flow diagram is used, which is also used in this study. A data flow diagram is interesting because the intention of the model is based around the data itself, which is highly relevant for privacy, and it is proven to have been used in a way for mapping out security risks, which is somewhat relevant for the study of privacy risks. In addition to this, LINDDUN Wuyts [2015] is a threat modeling methodology that does focus on privacy threats and also uses a Data flow diagram to find privacy threats.

6. Method

The method to study privacy design patterns in Firefox and Chromium Sync was set up in the VAF phase of the project. In short, the overall steps were to first recognize which privacy design patterns are relevant in the Sync functionality, and then identify the implementation of these patterns within these software systems. To do this, basic information about the system is needed, including information about the data and its flow throughout the software systems.

There were no automatic approaches to recover the privacy design patterns out of the existing documentation and code of Firefox and Chromium, which meant a manual approach was taken to both filter out the relevant privacy design patterns and to identify the implementation of these patterns. This approach was inspired by LINDDUN, STRIDE and a basic manual approach to gather information about a software system based on mapping out components in the code Biggerstaff [1989].

For this study, the privacy design patterns from privacypatterns.org in the categories Minimize, Hide, Separate and Abstract were chosen, which can be found in the Appendix. These patterns are for the most part concerned with privacy sensitive data, where the so-

[5] https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20)?redirectedfrom=MSDN

lutions can be implemented in the back-end of the code. This study was aimed at the documentation and source code of Firefox and Chromium Sync, and especially the back-end portion of the code. This was chosen because of the scope and time-constraints of the study itself.

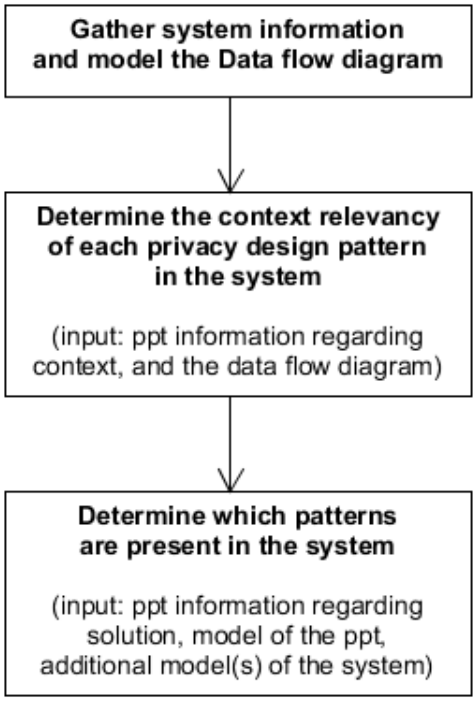

Figure 1: Overview of the method

The method included five steps: Step 1. Gather information about the system and model the DFD. The goal of the first step was to gain basic information about the system and its high-level components, which can be gathered from both the documentation and code. The result of this step was a Data flow diagram, which models the flow of data throughout the system and can be modeled at a fairly high abstraction level.

Step 2. Extract context related information from each privacy design pattern. The second step was concerned with studying the privacy design patterns themselves by studying the description as stated at privacypatterns.org. Based on the summary, context, problem, solution and examples, a interpretation of the privacy design pattern can be set up that can be used in conjunction with the results from step 1, in preparation for step 3.

Step 3. Determine for each privacy design pattern if the context of the pattern is possible in the user-data synchronization function of Firefox and Chromium. The goal of this step is to filter out the privacy design patterns that are not relevant in Firefox and Chromium Sync. To do this, more information might be needed about the software system to being able to conclude things about the pattern descriptions from step 3. The result of this step are the subset of privacy design patterns from the base list of patterns that might be found in the documentation and source code of the software systems that are studied, because these could have been implemented in theory.

Step 4. Extract further privacy design pattern information and model the privacy design pattern. The goal of this step was to organize the privacy design pattern information in a

way that it could be compared with the findings in the software systems. Because of the nature of the privacy design patterns it was not very helpful for the study to model the privacy design patterns because these patterns can be both very specific and abstract. The result of this is that each privacy design patterns is more of an interpretation and set of possible problems and solutions at a overview level, instead of an exact science which can be modeled with confidence.

Step 5. Determine which patterns are present within the user-data synchronization function of Firefox and Chromium. The goal of this step was to be able to say which privacy design patterns have been implemented in the software systems in question. Every previous step is used to create conclusions about the possible implementation and existing implementation of a privacy design pattern in the software system. Some type of risk assessment is important here as to being able to say something about the implementation or lack of implementation of privacy design patterns that were possible from step 3, and why the implementation or lack of implementation of the pattern is remarkable or not.

There was some deviation from the method because in practice it was found that the privacy design patterns are too abstract to model them efficiently and the information needed from the system was both time-intensive to gather and verify and a lot of detailed information is needed for a good conclusion about the implementation of a privacy design pattern in a software system. Because of this, the modeling of privacy design patterns in step 4 was not carried out, and step 5 of the method is more of an exploration about the identified privacy design patterns in this simple use case of Firefox and Chromium sync, instead of an exhaustive analysis with a lot of detail about how the privacy design patterns are or could have been implemented.

Finally, it was not determined how much detail was needed in the modeled information about the software systems under study; it has been found that the information about the data and the handling of this data is key and is therefore the focus in the study of the systems. This means that the focus of the study of the systems was on the architecture, the flow of data and its handling and storage. In this case of Sync this meant that the detailed implementation of Sync was not as important as the overall components and their behavior, because Sync is a simple storage and distribute functionality.

7. Firefox Sync

7.1. Analysis approach

This chapter is concerned with identifying the relevant privacy design patterns for Firefox Sync, and then identifying which ones could have been implemented and have been implemented. To analyze Firefox Sync in light of the privacy design patterns, both the documentation and source code have been studied to gather information about the system of Firefox Sync. Because of the scope, the privacy design patterns and the general architecture and use case of Firefox Sync have been studied as to filter out the relevant patterns. Information that was needed to make a basis for claims surrounding privacy design patterns and Sync was retrieved from said documentation and source code of Firefox Sync.

Firefox has extensive documentation surrounding Sync, but details and overall functionality had to be filled in and validated by investigating the source code in a manner suffient to draw conclusions surrounding the privacy design patterns. The source code is more secure in this manner, seeing as source code generally cannot lie and the documentation could be outdated or missing crucial elements.

The documentation was used for information surrounding the overall setup of Firefox Sync, an outline of the data structures used, the overall workflow of Sync surrounding user setup of the service and what type of encryption is used with the token service. Most of this documentation was however not sufficient or up to date with the source code to gain a clear enough picture about Firefox Sync, so the source code was investigated to fill in the gaps and to validate. The source code was used for validation by for instance going into the details about how the system actually packages Sync data, even though this was well documented. This validation was not possible for every facet of Sync seeing as the scope of this study focuses on the client-side code.

The assumptions surrounding Firefox Sync for this study are that the intention of the documentation and source code is executed flawlessly in Firefox Sync both at the client and the server side. The open source source code has been investigated, but not the server code. It is assumed that the server side does not do anything with the (encrypted) data, other than storing and distributing this Sync data. It is also assumed that there are no hidden backdoors in the way this authentication is set up, which would be able to threaten the privacy friendliness of the source code surrounding Firefox Sync.

7.2. Firefox Sync General

Firefox Sync

Firefox is a popular open source web-browser. Within Firefox there is a functionality called Firefox Sync. Firefox Sync allows the user to synchronize Firefox browser information between multiple user devices. Sync requires a Firefox account which is made possible by Firefox's Authentication protocol. If a user is registered he/she can choose to synchronize data like history, bookmarks and saved passwords, etc. The overall goal of Firefox Sync is to make the chosen browser related user data persistent in all the chosen user devices that use the Firefox browser and are compatible with Firefox Sync.

The browser data that can be synced is: History: all visited sites in Firefox; Bookmarks: all saved bookmarks in Firefox; Logins and Passwords: saved login credentials and passwords in Firefox; Open Tabs: the tabs in Firefox that are currently open for this user; Add-ons: Firefox browser specific add-ons that are compatible with Sync; Addresses: fill-in-

form related information that can be filled in automatically in Firefox; Options: the Firefox browser settings for this user that can be synced. The user can choose which data should be synced, which can include one or all of the above options.

THE USE CASE

When a user has set up Sync on one device [6], other devices should be coupled to make a sync possible. To do this, a user has to enable sync on another device with the same Firefox Sync credentials. To do this, a sync key has to be provided to couple the chosen user devices. This coupling process logic is handled by the Easy Setup service by Firefox. This Service handles the communication and setup of the user services using the Sync key to authenticate the devices. This sync key is transmitted behind the scenes to not over-complicate this process for the user. After this coupling process, Sync should be running behind the scenes on the client-side to check if a sync is possible for each individual user device connected to sync.

Firefox Sync stores the user data in a centralized way [7]. This follows a client-server approach, where the user device acts as the client and the Firefox Sync Server as the server. Firefox Sync works based on a client heavy approach because the server only stores the encrypted data and some non-encrypted meta-data and the client side therefore does almost all the heavy lifting for sync. The packages that are being transmitted between client and server are universal data sets called WBO's (Weave Basic Object). The client side provides the main functionality with these data sets by using the server as a storage server to get and push the latest user data that should be synced.+

TECHNICAL DETAILS

For this project we are looking into the Firefox Sync code and the way data is handled regarding the privacy design patterns. The client-side for Sync is mainly written in C++ and Rust, and the rest of the functionality of Sync uses the browser functionality on the client-side that is already in place for storing sync related data, like bookmarks. The server side handles the WBO data sets and its meta-data in JSON. The client-server communication uses HTTPS.

7.3. THE DATA AND ITS RISKS

The data that is shared between devices in Firefox Sync is personal to the user and has a certain risk associated with the possible leaking of this data. Each type of data will be discussed here to get an overview of the possible risks associated with the leaking of the data outside of Firefox Sync.

ATTACKERS

An attacker in the Sync scenario is someone who tries to get data and/or information about the service that is linked to a person or a group. Data that can be associated with an identifiable person is most at risk because this can be transformed into information about a person or a group, which is advantageous to have for the attacker. An attacker could use this information to perform other attacks, for instance, identity fraud, phishing or to simply build up a profile of someone and sell this to other parties. Most data that is handled

[6] https://mozilla-services.readthedocs.io/en/latest/sync/lifeofasync.html
[7] https://mozilla-services.readthedocs.io/en/latest/sync/overview.html

by Sync is vulnerable in an unencrypted form and could be identified to the e-mail address that is used.

If for instance an attacker would be able to gain information about a person with the users history and bookmarks, a profile can be set up by an attacker to then try and gain access to passwords for other services that the user might used with known email addresses. History and bookmarks can give an attacker a wealth of information of for instance: interests, occupation, vacation ideas, maybe location data, purchases and possible future purchases, etc. With this information a profile can be crafted by an attacker that can then try and guess passwords on other services because enough personal information is known to try and guess passwords with words and concepts that are familiair to the user and thus might be more likely used as a password.

THE DATA

The Sync data is very interesting to attackers when it can be accessed in a complete form and linked to a person: [8] [9]

History and Bookmarks:

Both the history and the bookmarks are a history of web addresses that were visited. Bookmarks can also hint to the intention as of why the web-address was visited in the first place, for instance, a bookmark of a product from a on-line shop could be an indication of someone's interest in buying said product. Someone's history is vulnerable for quite obvious reasons because it might indicate very personal information like health condition information, but also someone's background, demographic, interests and occupation, etc. The more information that is saved over time the more this can say about a person when it is linked to this person. This makes both the history and the bookmark information very vulnerable in the case that it should leak.

Logins and Passwords:

A leakage of login credential and password information is disastrous for a user or a company because in this way accounts can be accessed that shouldn't be, accounts can be taken over and blocked, personal information apart from the browser data could be accessed, etc.

Open tabs:

If the open tabs of a user should leak this could give information about what a user is doing right now and could say a lot about a person over time like in the case of history and bookmarks. Just imagine the case where an attacker is looking over the shoulder of the user. This piece of information is very interesting to an attacker because of its current time sensitivity and could be used to leverage phising attacks or other types of fraud.

Add-ons:

Add-on information could give information about what type of interest the user has of his/her preferences in a on-line browsing environment. Some add-ons are merely functional, like for instance, ad-blockers; others could be very specific, like a Facebook add-on, which implies the usage of Facebook.

Addresses:

Addresses is comprised of all the form related data that is filled in automatically. This could include a full name, location and address, phone number, etc. It is convenient for a user to have this filled in automatically but it is very personal and is a possible point

[8] https://mozilla-services.readthedocs.io/en/latest/sync/objectformats.html
[9] https://mozilla-services.readthedocs.io/en/latest/sync/storageformat5.html

where an attacker can link the other browser data to a specific person, which is a worst-case scenario.

Options:

Options are comprised of preferences of the Firefox browser itself. This could indicate the tech savvyness or the privacy habits of a user. It could indicate the search bar that is used, which could be valuable information to an attacker. Options is possibly the least harmful of the categories, but when it comes to information gathering, usually more is better.

Related attacks

There are several points of attack for an attacker. The first one is the client-side, which is the personal computer of the user, the second is the connection between client and server and the third is the server itself. Gaining access to the client-side makes attacking Firefox Sync obsolete because in this case all the data is already vulnerable. The main point of the attack would possibly be in the other categories.

For the connection between client-server, if there is no secure connection the data packets could be monitored and inspected by a possible attacker. There could also be a man in the middle attack which would modify the data exchange scenario in a way that is advantageous to the attacker and not the data subject. In Firefox Sync however, the connection is secured with HTTPS and a basic authorization scheme that is required to communicate with the server. It is assumed in this study that HTTPS works as intended and is secure, which lowers the risk of all related attacks that try to monitor or alter the data stream between client and server.

The server side could be attacked as to gain access to the data that is contained at the server side. A possible angle is for an attacker to act as though it is a legitimate client, but this is difficult because of HTTPS and the authentication that is enforced. Another attack is to gain access in a legitimate way, as an admin for instance, and gather the data from multiple users and then reuse this data for other purposes. This is difficult because the data is always encrypted in Firefox sync and can only be decrypted by the client using Firefox Accounts credentials, where the decryption key is different for each user and its subsequent data. An admin should not be able to gain access to the information that is needed to be able to decrypt the user data, which is feasible and logical in a storage and distribute application where no manual operations are needed.

Conclusions

The use case for Firefox Sync is a simple store and distribute between client and server. The data is however very vulnerable in its unencrypted state because it is easily linked to a user even without the specific client and user information. Therefore the possible damage of the data in this form leaking, for thousands of users, is very high. Luckily, the data is encrypted encyptred before it is sent to the server and only decrypted at the client side. The only visible data is the meta-data, which is still protected by a secure HTTPS connection.

The points of attack are mainly focused on the connection between client and server and the server itself. Here, the data and implementation to secure the data is analyzed at a very high level, a more detailed discussion of the risks and possible privacy faults is given in chapter 4.5.

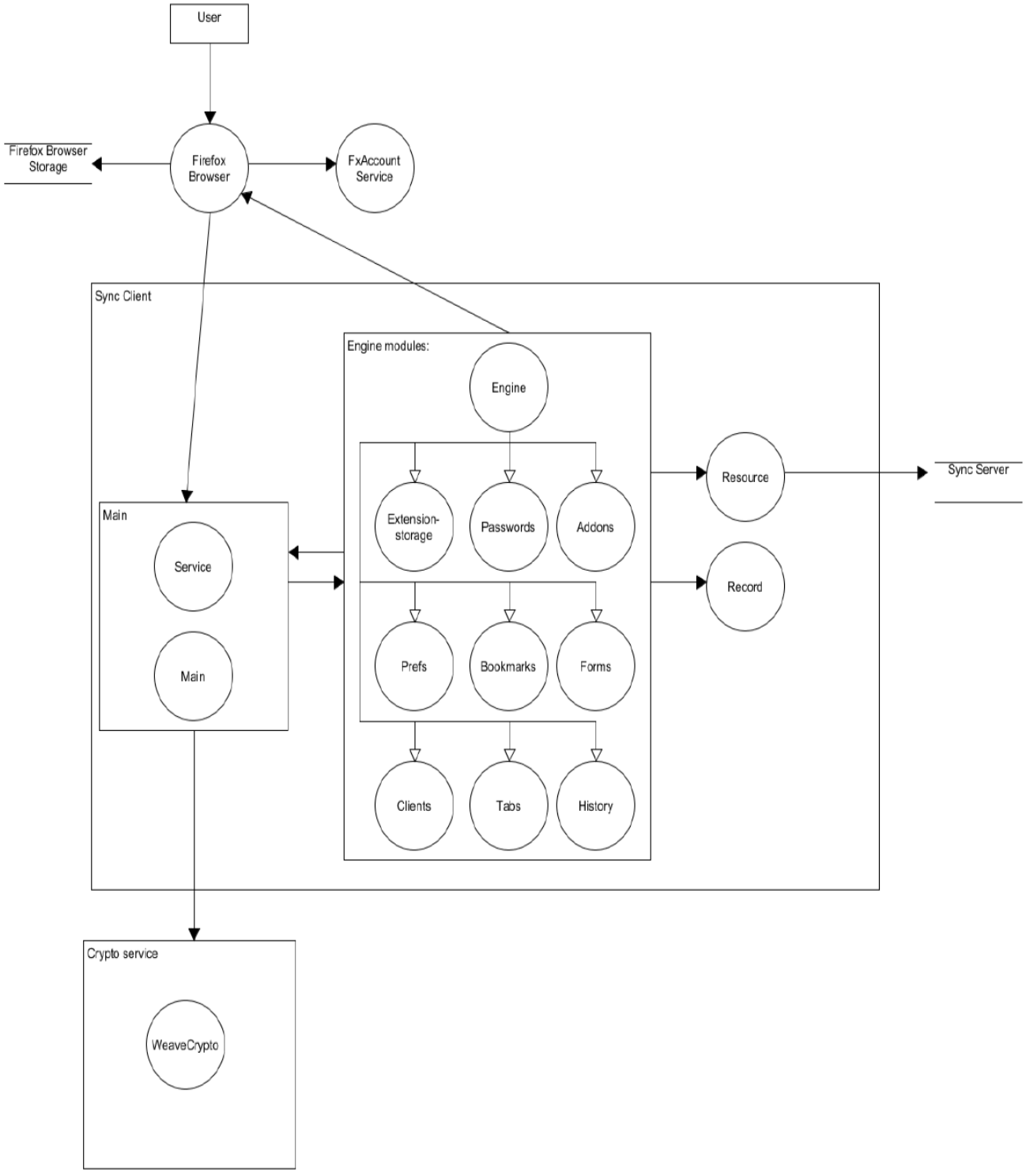

Figure 2: Firefox Sync Data flow diagram

7.4. DATA FLOW DIAGRAM FIREFOX SYNC

The main components of Sync are the client-server implementation and the specific handling and storing of the data that is shared between client and server. The browser data is stored at the client-side in a typical manner for the Firefox browser. This data and its implementation in the code for retrieving and storing the data is then leveraged to gain access to the data for Firefox Sync. The data that is new for this client is checked by Sync by using the meta-data on the server, which is then downloaded on the server and stored back into the Firefox browser storage. The Sync component determines what data should be synced and how it should be synced and stored at the client and server side. The sync server simply stores this (encrypted) data. Finally, this is all made possible because of Firefox accounts, which enables tokens to be made that are sent from a token server, which makes identifiying the relevant data for Sync and the client possible.

The client-side is composed of a few different components to do this (see Figure 2). It has a basic component named Service[10] that can set up Sync for this client and triggers the other components used for Sync that are initialized. Each data type that is possible in Sync, for instance bookmarks, has its own engine [11] and implementation to retrieve the data from the browser and server, package it and send it off to the component that handles the HTTPS communication and encryption named Resource.

These engines include all the data types that are recognisable from the data, plus some more used to make Sync work, like for instance client info. Engines leverage existing implementations to store Firefox browser data that is transformed into a Weave Basic Object [12] that can be stored on the Sync server. In this way the Engines are connected back to the Firefox Browser, but are observers of the main Sync Service.

The components that handle the server side communication can handle the Weave Basic Object format and send this to the server via basic HTTP commands that are then sent via HTTPS. This is an universal data format for Sync data which can fit all the different types of Sync data.This is handled in the Record component (for Weave Basic Objects).

Before sending data off to the server it is first encrypted using AES, and it is decrypted when it is retrieved from the server so that the new data can be stored on the Firefox browser side on the client-side. WeaveCrypto is used to handle this encryption [13], which is used by the main Sync Service component.

In this way the use case surrounding Firefox Sync is not very complex because there is only a mapping between Firefox browser stored data like bookmarks, that are then retrieved and transformed into a WBO that is encrypted and then sent to the server. The client-side handles all the logic as of what, when and how to sync, which is quite complex but not really relevant for a discussion around data in this way except for what mutations happen to the data, which seem to be none (except for the mapping). Firefox Sync is a simple packaging and distributing of data from client to server and back to client.

The weave basic object is generated in the Firefox Sync functionality itself, using the engines and the Record class to populate the right data. This then gets sent to the server. The weave basic object is thus utilized in the Sync functionality and at the server side only,

[10] https://searchfox.org/mozilla-central/source/services/sync/modules/service.js
[11] https://searchfox.org/mozilla-central/source/services/sync/modules/engines/bookmarks.js
[12] https://searchfox.org/mozilla-central/source/services/sync/modules/record.js
[13] https://searchfox.org/mozilla-central/source/services/crypto/modules/WeaveCrypto.js

while the browser functionality of Firefox does not use this weave basic object but only the browser data.

According to the Data flow diagram then, the data flows from the Firefox browser to the Sync functionality and back, which transforms the data into Weave basic objects that can be sent to the server and back. The payload of the different data type objects itself might be different, which is illustrated in the different engines that are active, but is not relevant for this discussion but only in the back-end implementation because the overall data structure that is sent to the server remains the same.

DATA

One of the most important aspects of this high level view of the function is how the data and meta-data is packaged and stored. For Firefox Sync a Weave basic object has the following fields:

- id
- payload:
- cipher-text
- IV
- hmac
- modified

The payload contains a cipher-text, IV and hmac, which contain the base64 encoding of the fields that are used for this message. The cipher-text contains the encrypted data, which has a different structure for each data type. This is however not that interesting to discuss here, seeing as this is encrypted data and is handled on the client-side. The modified field gives an indication as of when the data was modified last.

The clear-text that is derived from the cipher-text is a JSON string representing an object.

META-DATA

The server needs some kind of meta-data so that the client knows which data is needed for this Sync client and which fields need to be updated. This is because the server does not do anything other than store data and react to HTTP calls.

Only a meta-global record [14] remains unecrypted at all times. This meta-global record contains a storageVersion (with the version of storage used for Sync), a syncID (an opaque string that changes with changes to the overall data), and information about the different engines used (a hash with fields of the engine names and values). Data concerning the engines in use contain a syncId field and a storageversion. This data therefore stores the engines that are in use for this client and are visible on the server side. If this meta-global record is studied over time it should indicate if the overall data is changed, by looking at the SyncId, and which engines are active and have changed over time by tracking each engine's SyncId.

[14] https://searchfox.org/mozilla-central/source/services/sync/modules/telemetry.js

This meta-data is used to point to the specific data for each datatype, which is used for retrieving the right datatypes from the server by the client.

For the client and server to know which data is for this user by using an ID field that is generated at random and is not linked to the user in a direct way, and a token server is used to distribute tokens that can be assigned to after authentication on the client side. In this way the id on the server-side can be opaque and the sets of data by itself cannot be traced back to a user.

From the outside looking in, it is not clear which data sets contain what kind of information, as long as you don't possess the decryption keys that could decrypt the cipher-text The id's of a WBO can be tied back to the engine it belongs to via the meta-data of this global record. The client needs to know this as well to make Sync possible. If the database would leak on the server side, you could therefore point out which data sets belong to which engine, but not its contents.

DATA TYPES

The data types are similar to the data that can be synced for a user which have been discussed above. The data types that are sent back and forth from client to server are:

- Add-ons
- Bookmarks
- Clients

  This set contains the relevant client-information that is used for Sync, which is also encrypted. This contains OS information, an unambiguous identifier for the client app, which kind of browser version is used, what type of device (laptop, tablet, etc) and device hardware info (which type of model, only mobile)
- Commands

  These contain the commands/processes that need to be distributed between clients to make Sync work. For instance, resetEngine.
- Forms
- History
- Passwords
- Preferences
- Tabs
- Crypto/Keys

  The keys that can decrypt the individual encrypted data sets are also sent to the server as a record. To decrypt this record the master key is needed, which is provided via the a token server and uses Firefox Accounts. These keys are bundled together and are derived via SHA-256 HMAC-based HKDF [15]. This field contains the default keypair with encryption key and hmac key.

[15] https://mozilla-services.readthedocs.io/en/latest/sync/storageformat5.html

CONCLUSION

Based on this high-level information a few observations can be made about the handling of data throughout the functionality of Firefox Sync. The first is that the data is only being transformed into a WBO to the server, and is being converted back into browser related data for the Firefox browser as to update the client. The second is that the client-side handles all the functionality except for storing and fetching the original browser data on the client-side, and storing the data on the server. The third is that the data is encrypted before it leaves the client, which means that the data is always under the control of the client. The meta-data that is transmitted and stored on the server in an unencrypted form is not a great risk for the user in this functionality because it does not contain information that can be traced back to the user, nor does it contain a lot of information by itself.

7.5. CONTEXT-RELEVANT PRIVACY DESIGN PATTERNS

The first context-relevant privacy design patterns will be discussed here based on the general information regarding Firefox Sync as provided in the previous chapter. This type of discussion is possible because the privacy design patterns can be at the same time general and specific to certain use cases. An example is the anonymous reputation-based blacklisting pattern, which is only possible of being implemented in a messaging scenario where multiple users are anonymous but need to be blacklisted to prevent troublemakers from using the service, without breaking anonymity. This pattern is of course not relevant in the scenario of Firefox Sync because this pattern has clear preconditions that are absent in the sync use case. This chapter will thus list all privacy design patterns chosen for this study and determine based on the privacy design pattern context and problem descriptions which privacy design patterns are already not relevant in the context of Firefox Sync.

The complete list of the privacy design patterns is provided in the Appendix.

PRIVACY DESIGN PATTERN FILTERING

PPT1: Strip invisible meta-data

This pattern wants to remove possibly sensitive meta-data in applications where the user provides data that might include meta-data that is not known to the user to be present. An example would be to upload an image to a service which could include information about when the picture was taken or maybe even where it was taken. The description of this pattern is as follows: Strip potentially sensitive meta-data that isn't directly visible to the end user. [16] Context: The context is not relevant for Firefox Sync because in this pattern the implementation revolves around removing meta-data that is transmitted by the user in a functionality. For Sync it is therefore not applicable because the user does not upload images or documents themselves, but all the sync data that is transferred to Sync is already available in the Firefox browser. The data that does get transmitted to Firefox Sync in this scenario to Sync is encrypted and is not accessible to other parties that might use or share information stored in the meta-data that could be transmitted with Sync.

PPT2: Added-noise measurement obfuscation

This pattern aims to add noise to a measurement or other data that is needed to provide a service, as to obfuscate the contents while still being able to use this data as input for the service. The description for this pattern is as follows: Add some noise to service operation

[16] https://privacypatterns.org/patterns/Strip-invisible-metadata

measurements, but make it cancel itself in the long-term [17] Context: This pattern is not relevant for Firefox Sync because this pattern relies on a value or set of values that can be manipulated as to have the attacker not know the real value that is being sent back and forth. In this pattern there is also a possible loss of data, which would make Sync not work properly because the encrypted complex data that is being sent to the server can probably not handle noise obfuscation for decryption to work.

PPT3: Location Granularity

This pattern aims to make location data more granular and privacy friendly by identifying which granularity level of the location data is needed. An example would be to track someone's general location, i.e. a city, instead of the exact location. The description is: Support minimization of data collection and distribution. Important when a service is collecting location data from or about a user, or transmitting location data about a user to a third-party. [18] Context: This specific pattern is not relevant for Firefox Sync because there is no location data transmitted or stored in Firefox Sync. The data-sets that are being sent cannot be made more granular by itself because it is a complex data structure and not a specific value that could still be useful in a functionality like location data.

PPT4: Personal data store

Personal data store tries to retain the control of the user over his/her data by implementing a personal token, database system and local web service, for instance, and to specify which data is shared with a central server. An example would be to distribute medical data to a server that is needed to provide a service, and to keep the rest of the data running on the local service. The descrpition is: Subjects keep control on their personal data that are stored on a personal device. [19] Context: This pattern is not relevant for Firefox Sync because users never lose control over their data in the case of Synchronization. The data that is stored on the server is encrypted and is only used to distribute it back to the user itself. Also, the Firefox Sync data is data that was already available in the Firefox browser.

PPT5: User-data confinement pattern

The user-data confinement pattern tries to shift processing of data to user owned devices instead of entities not controlled by the user. The description is: Avoid the central collection of personal data by shifting some amount of the processing of personal data to the user-trusted environments (e.g. their own devices). Allow users to control the exact data that shares with service providers [20] Context: This pattern is not really relevant for Firefox Sync because the context and problem of the pattern is not exactly applicable. The context states that "This pattern may be used whenever the collection of personal data with one specific and legitimate purpose still pose a relevant level of threat to the user's privacy". The problem states that users have to trust systems that store data in single central entities, and the sharing of personal data is necessary to do this. The goal of the pattern is to shift some or all of the processing of the data to the user-trusted environment. In the case of Firefox Sync the server does not process the data at all, thus it is not relevant.

PPT6: Anonymous reputation-based blacklisting

Anonymous reputation-based blacklisting tries to keep the anonimity of users, in for instance an online forum, while still being able to ban these anonymous users. The de-

[17] https://privacypatterns.org/patterns/Added-noise-measurement-obfuscation
[18] https://privacypatterns.org/patterns/Location-granularity
[19] https://privacypatterns.org/patterns/Personal-data-store
[20] https://privacypatterns.org/patterns/User-data-confinement-pattern

scription: Get rid of troublemakers without even knowing who they are. [21] Context: This pattern is not relevant for Firefox Sync because it there are no anonymous users nor is there a use case where banning a user is relevant.

PPT7: Encryption with user-managed keys

Encryption with user-managed keys tries to have the user keep control over the means of encrypting and decrypting data, and handle the service in a way as to not enable the service provider to encrypt or decrypt the users data. The description: Use encryption in such a way that the service provider cannot decrypt the user's information because the user manages the keys. Enable encryption, with user-managed encryption keys, to protect the confidentiality of personal information that may be transferred or stored by an untrusted 3rd party.[22] Context: This pattern is relevant in Firefox Sync because there is a server side which stores data, and this data could be vulnerable without encryption on a server (even Firefox's servers) or when the keys are also stored and can be decrypted by Firefox.

PPT8: Use of dummies

The use of dummies pattern tries to obfuscate user actions so a possible attacker that tries to gain information from said user actions cannot distinguish these fake actions from real, hindering the information gathering and attack process. The description: This pattern hides the actions taken by a user by adding fake actions that are indistinguishable from real. [23] Context: This pattern is relevant for Firefox Sync because it could be interpreted that a user triggers an action by enabling Sync, even if the action originates in the Sync Client, which then leads to a scenario where dummy actions could be relevant to try and hide what data is sent to the server and back.

PPT9: Pseudonymous messaging

The pseudonymous messaging pattern tries to implement pseudonyms for users by using a trusted third party in a messaging scenario, decoupling sender from receiver. An example would be anything using online interactions, for instance, an online forum, where the users cannot be identified as real persons by using a pseudonym used by the server. The description: A messaging service is enhanced by using a trusted third party to exchange the identifiers of the communication partners by pseudonyms. [24] Context: This pattern is not relevant for Firefox Sync because this is not a messaging service where users can interact and a pseudonym would be relevant.

PPT10: Onion routing

The onion routing pattern works by sending and encrypting/decrypting data between different nodes that are loosely coupled, decoupling the first sender with the eventual receiver and vice versa. This can be used in situations where data is sent back and forth, if a node network is implemented. The description: This pattern provides unlink-ability between senders and receivers by encapsulating the data in different layers of encryption, limiting the knowledge of each node along the delivery path. [25] Context: This pattern is applicable in Firefox Sync because in theory, the information from client to server could be relayed via multiple nodes before it reaches the end server of Firefox Sync.

PPT11: Pseudonymous identity

[21] https://privacypatterns.org/patterns/Anonymous-reputation-based-blacklisting
[22] https://privacypatterns.org/patterns/Encryption-user-managed-keys
[23] https://privacypatterns.org/patterns/Use-of-dummies
[24] https://privacypatterns.org/patterns/Pseudonymous-messaging
[25] https://privacypatterns.org/patterns/Onion-routing

The pseudonymous identity pattern tries to couple users to pseudonyms that are not retracable to the users themselves. This can also be used in online forums or email scenario's. The description: Hide the identity by using a pseudonym and ensure a pseudonymous identity that can not be linked with a real identity during on-line interactions. [26] Context: This pattern is not relevant for Firefox Sync because users are not identified by public identities, as the pattern Context text states. There is also not interaction with other users, even anonymously in Firefox Sync.

PPT12: Aggregation gateway

The aggregation gateway pattern tries to implement a way to work with an aggregated encypted data load for users while still having a functional service. This uses multiple parties that should be trusted or not get direct access to the data. The description: Encrypt, aggregate and decrypt at different places. [27] Context: This pattern is not relevant for Firefox Sync because there is no load that can be aggregated in this use case, which has to be monitored to make the functionality work. The data that is transmitted is always tied to the user itself and should not be grouped together with other users because the data is too much of a set and other users do not need the same demands placed on their data time-wise, seeing as users can range in when they are using Firefox and thus Firefox Sync.

PPT13: Trustworthy privacy plug-in

The trustworth privacy plug-in tries to implement a way for user data to get sent to a server in a way that is aggregated and generated at the client side. An example is the smart meter that might get data that is processed in an aggregated way at the users side and is sent to the server for billing purposes on a monthly basis, not data being sent on an hourly basis and processed on the server side. The description: Aggregate usage records at the user side in a trustworthy manner. [28] Context: This pattern is not relevant for Firefox Sync, because usage records are not being tracked explicitly and are not processed per se to make Sync work. Also, for this service to work detailed information is needed on what to synchronize and cannot rely on abstracted data that is never inspected individually as a value.

PPT14: Anonymity set

The anonymity set aggregates data into a set as to lose a bit of granularity, making the data more privacy friendly. For instance, if the entity needed is daily consumption of a service, this can be tracked by using user input that is stripped of all features that make a user profile a user profile, except for the minutes the user has used the service this day. The description: This pattern aggregates multiple entities into a set, such that they cannot be distinguished anymore. [29] Context: This pattern is relevant for Firefox Sync because the data that is stored in a WBO can be aggregated further for a user and the functionality would still work.

UPDATED LIST

In conclusion, the list of privacy design patterns that are left after the filtering of patterns based on the general sync use case are as follows:

- Onion routing
- Use of dummies

[26] https://privacypatterns.org/patterns/Pseudonymous-identity
[27] https://privacypatterns.org/patterns/Aggregation-gateway
[28] https://privacypatterns.org/patterns/Trustworthy-privacy-plugin
[29] https://privacypatterns.org/patterns/Anonymity-set

- Anonymity set
- Encryption with user-managed keys

These patterns will be studied further in the next chapter in light of Firefox Sync by setting up a deeper more detailed discussion of Firefox Sync and the context and risk-factor of each privacy design pattern.

7.6. Observed Implemented Privacy design patterns in Firefox Sync

Onion routing

Is it implemented? No.

Onion routing basically means that data is routed between different nodes. In Firefox Sync there is a direct connection between a client and a server using HTTPS, and each client is assigned a specific server when Sync is first enabled. There seems to be no evidence therefore of using Onion routing to make the traffic from client to server more secure, nor would it be a logical solution for Firefox to host its own Onion node network.

Is it a risk/privacy fault?

It is not a great risk and therefore not a privacy fault per se, seeing as the data is encrypted when it travels from client to server and back, and this data transfer is executed via a HTTPS connection.

Implementing Onion routing for Firefox Sync would make the transfer of data more secure but even when HTTPS fails, the only real risk is that the meta-data is revealed and linked to a specific domain. This meta-data does not reveal the privacy risk data itself but only what types of data this user has enabled, how frequently the data types in general are updated, and when the user is using Firefox Sync. Breaching HTTPS would be a very dangerous but infeasible situation, therefore the risk is low. The data that could be revealed when it is linked to an individual is based on the meta-data that is unencrypted, which is not a great privacy risk by itself.

Use of dummies

Is it implemented? No.

Use of dummies is hiding user actions by adding fake actions. A fake action in the case of Firefox Sync would be a server check for new updates, or a synchronization itself. There is no evidence in the documentation or source code that fake actions are inserted with the intention of hiding either the usage of the synchronization functionality with server update checks for a specific user, or when a complete synchronization is transferred.

Based on the context description of the pattern itself, this is more of a last effort solution when the data cannot be obfuscated in other ways. Nevertheless, this pattern could have been implemented as to hide synchronization actions in the case where an attacker is capable of monitoring the usage of Firefox Sync for a specific user. The implementation of this pattern would in this case strengthen the privacy of the use case.

Is it a risk/privacy fault?

It is not a great risk, and other implementations of patterns and best practices, like HTTPS and encryption on the client-side and no decryption on the server side, already lower the risk and impact that would be mitigated by using this pattern. It is probable that someone monitoring the usage of Firefox Sync of a user could see how much data is going to what domain, and could figure out that the destination is Firefox Sync. However, this

does not say much about a user itself, except for the usage of Firefox Sync itself. Hiding this behavior with dummy actions could help, but is not a great risk with a lot of damage to the user.

Anonymity set

Is it implemented?

Anonymity set means that entities cannot be distinguished to specific users, as to make analyzing a users activity feasible. In the Firefox Sync scenario this is the case at the server side because the client-side retrieves the right information from the server using the Firefox accounts information, and the id's on the server are opaque. In a scenario where a users activity is already being monitored, and all packets of data are sent back and forth to the client and server, the different types of Sync data can be distinguished from each other in the meta-data information.

An interpretation of this pattern could mean to make data structures more abstract as to make them more privacy friendly, and this could have been implemented more in Firefox Sync. For instance, bundling all the Sync data together in one data set, that then has to be unpacked at the client-side and checked for new updates. This would hide meta-data about what kind of Sync data types are enabled for the user and when they were last updated, which would make the function more privacy friendly if monitoring the sync meta-data and linking it to an individual user is feasible.

Is it a risk/privacy fault?

It is not great privacy risk and therefore not a privacy fault per se because even if privacy could be enhanced by hiding more data by packaging it in a bigger set, the usefulness of this pattern in this scenario assumes that the meta-data can be inspected and is linked to an individual user. This would mean that the opaque id of the meta-data set is linked to an individual user, and the meta-data on the server is leaked for this specific user.

Encryption with user-managed keys

Is it implemented? Yes

Encryption with user-managed keys means that the user manages the encryption keys and the service provider cannot decrypt the data. The user needs to generate and manage the master key themselves. This pattern has been implemented except for the generation of the master key and explicit management of this master key. All data that is sent to the Firefox Sync server has to be encrypted, except for the meta-data that is not linked directly to a user but to a Firefox Accounts session token. The master key is generated based on a password the user provides in Firefox Accounts. This master key is stored in the Firefox Accounts portion of the software and never goes to the Firefox Sync servers. In the Firefox Sync code there seems to be no evidence of Firefox trying to retrace the pass phrase nor of sending the encrypted pass phrase to the Firefox Sync servers for storage.

The Sync functionality was set up with the intention of Firefox not being able to retrace the pass phrase or master key for Sync, nor being able to do anything with the user stored data on the Firefox Sync servers because of its encryption. The encrypted password is managed by Firefox Accounts however.

In the Firefox Sync functionality you would need the users password to generate the master key for Firefox Sync, which would then be able to decrypt the keys that are used to encrypt the Firefox Sync data. Because the password is not sent directly to the Firefox Sync servers, the only logical point of attack would be to acquire the users password. This does

however not compromise the privacy pattern in question, which still keeps the control of encryption at the user side, although the Sync master key is not generated nor managed purely by the user itself but through the users Firefox Account password.

For the Sync functionality the main functionality is to distribute already existing data to other clients. To encrypt the data before it gets distributed means in this case that the connection between client and server has less risk, and the server cannot process the data and therefore has less risk for the users privacy. This pattern alone in this fairly simple functionality removes a lot of risk and focuses the point of attack on the client-side, gaining the users password for Firefox Accounts and Firefox Accounts itself, but leaves Firefox Sync in a low risk state.

7.7. Conclusion

Out of the four feasible privacy design patterns that could have been implemented in the context of Firefox Sync, the principles of one privacy design pattern is implemented within Firefox Sync. Largely because of this pattern, a lot of risks are mitigated which makes the non-implementation of the other three possible privacy design patterns not as severe of a risk within Firefox Sync. The scope of the study in combination with the scope of the functionality of Firefox Sync concludes that these four patterns could have been implemented (more), and that the evidence of implemented privacy design patterns within Firefox Sync is as concluded.

8. Chromium Sync

8.1. Analysis approach

The analysis approach is the same as for Firefox Sync, as to have a second go at the approach regarding the identification of privacy design patterns in a client-server setting. Chromium had almost no documentation surrounding its functionality, architecture or general approach surrounding Sync, but has open-source source code. The source code does provide extensive comments, which was useful in the analysis. This source code is searchable and can be found at https://source.chromium.org/chromium/chromium/src

The assumptions surrounding Chromium Sync for this study are that the intention of the documentation and source code is executed flawlessly in Chromium Sync both at the client and the server side. The server code is managed by Google and is closed source, so this was not part of the study. It cannot be deduced what the server does with the incoming Sync data, but it is at least assumed that the server distributes the data to the client and back in a way that makes Chromium Sync possible. It is also assumed that there are no hidden backdoors at the Chromium Sync client-side.

8.2. Chromium Sync General

Chromium Sync

Chromium had a browser data synchronization function similar to Firefox Sync. This functionality ensures that Chromium related browser data got distributed from the Chromium browser client to all Chromium browser clients from this user that are connected to Chromium Sync. The basic use case is therefore similar to Firefox Sync, although the distributing of data is executed via the Google API and Google servers that store this data and communicate back to the Chromium Sync clients.

The data that could be synced was the same as was available in Google Chrome: Apps, Bookmarks, Extensions, History, Settings, Themes, Reading list, Tabs, Passwords, Addresses and Payment methods.

It should be noted that Chromium has almost no documentation surrounding Chromium Sync, and most of the documentation is illustrated in the different classes and methods in the source code itself as comments.

The use case

To set up Chromium Sync you need a Google account. The Chromium Sync client will then determine, similar to Firefox Sync, w which data should be synchronized or not. This means that the implementation of Sync is also a client heavy approach. The Google API seems to be only used to store the synchronized browser data and the data is encrypted before it leaves the client-side, although it exposes more meta-data than Firefox and it cannot be verified what the Google servers do with this data.

Technical details

Chromium is written in C++. Chromium Sync leverages the browser data storage capabilities of Chromium but has to translate it into a Sync data object. The connection with the Google API is HTTPS. Chromium also uses a key-store server to store its keys. AES is used to encrypt data, using a users pass phrase or the users password.

The data

Chromium has the following data items that can be synchronized with other Chromium clients via a Google account:

Bookmarks / Reading list

Bookmarks are similar to Firefox and any browser, for more information see chapter 4. Reading list is similar to bookmarks, as in, it is a feature that remembers links to URLs that could be read in the future.

Settings, Passwords, Open tabs, History, Extensions

These are similar to Firefox's syncable objects, see chapter 4.

Addresses (auto fill)

This is similar to general auto fill information that gets saved by the browser. This is often very personal information like e-mail address, phone numbers, home address, etc.

Payment methods (credit card)

This includes payment information that can then be conveniently synced to other Chromium devices. This is of course critical information seeing as acquiring this type of data is a first step for an attacker to being able to use the payment methods themselves.

Themes

This is theme based information for the browser for aesthetic purposes. This is shared between clients to share the same look and feel for the browser, but is not critical privacy wise probably.

Related attacks

First of all, the related attacks are similar to Firefox, see chapter 4. There is however more information that is gathered, like for instance a reading list in addition to bookmarks, and most importantly payment information. There is also another element of attack, which is a connection with a 3th party server and storage facility, from the point of view of Chromium, which are the Google servers for Sync.

There is more risk by using a third party way of storing the data because there is no way to monitor what happens on this side of the software. In addition, if some party managed to mimic the Google API and servers it could in theory be the case that data gets sent to the wrong party, which includes meta-data that is exposed. In the case of Firefox Sync the server is a storage bucket that can only store and send data, in the case of Google's server it is not known what happens with the meta-data, which leaves the risk on the server side higher than in Firefox's case. It has to be noted however that the intention has been by Chromium to not freely give too much information to Google, like by for instance hashing a syncId.

8.3. Data flow diagram Chromium Sync

Chromium Sync

Chromium has a separate Sync implementation apart from its browser features. This utilises Google for authentication (as a Google account is needed to use Sync) and is therefore linked to the Google API in this way. It is also linked to the usual Chromium storage implementation, where it also stores the for Sync in progress data and meta-data at the clientside. The rest of the Sync data is distributed via the Google servers.

A simplified version is shown in the DFD, see Figure 3. The user is linked to the browser and indirectly to the Google API when he/she is logged into Chromium with his/her Google

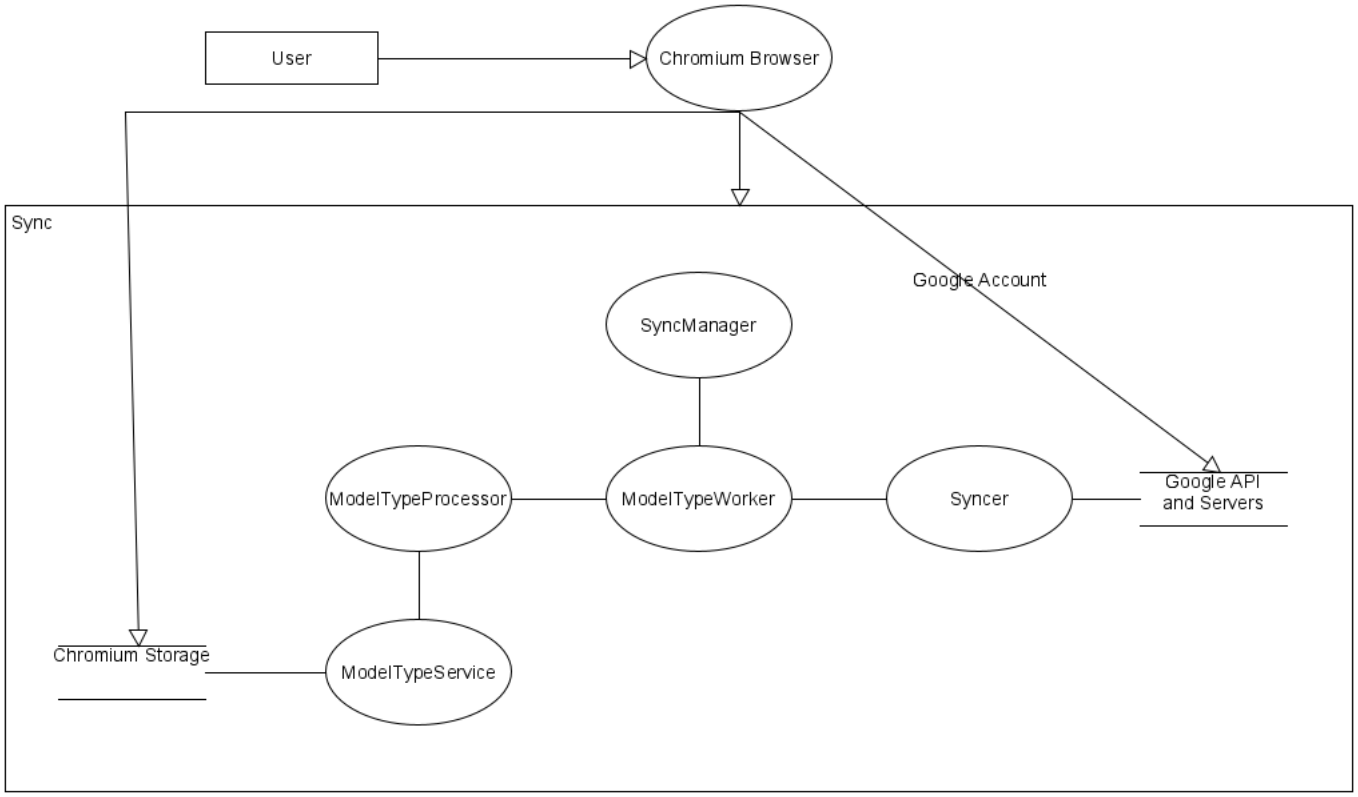

Figure 3: Chromium Data flow diagram

account. The browser is linked to the Sync implementation. Everything apart from the Google API and server is located at the clientside of Chromium.

To handle the normal Chromium data and Sync data, the ModelTypeProcessor [30] and ModelTypeService is used. [31] These communicatie with the Chromium storage for both the normal data and the Sync data and translate this accordingly. These components also manage the data and meta-data at this level of implementation.

The Sync related data is therefore populated by the ModelTypeProcessor and used from this point onwards as Sync data to the server and back to the internal storage of Chromium. The lifecycle of such sync data is therefore pretty broad , but is only used at the internal Chormium storage side as caching requires.

The SyncManager and ModelTypeWorker are part of the Sync engine. This component is concerned with the overall logic for Sync and includes the listeners and observer logic for the Chromium browser concerning Sync. The engine determines what and when data needs to be synced, and handles the communication with the ModelType side of Sync accordingly. The encryption part of the data is also located in the engine component and uses AES, similar to Firefox.

The Syncer component is the bridge between the server and Sync and handles the creation of commit messages to the Google servers.

The biggest part of the functionality is to track which datatypes need to be synced and handling this at the clientside for Sync. This is similar to Firefox. The real implementation is a lot more complex but does actually handle the data like it is specified in the data and meta-data sections of the code.

Data gets encrypted before it gets send to the server and the SyncId is a hash, as in, the server does not know what to do with this or how to use it. The data gets translated from and into Sync datatypes and the meta-data gets stored to make Sync possible. There does not seem to be any more handling of data apart from the core functionality. The high-level architecture is fairly simple and the main concern of the data is how it is structured, because this also says a bit about how it is stored and how it could be (mis)used.

DATA

The data types within Chromium are [32]:

- Bookmarks
- Preferences
- Passwords
- Auto fill profile information
- Auto fill (credit card and address information)
- Auto fill wallet meta data (usage counts and last use dates from Iwallet cards and addresses)

[30] https://source.chromium.org/chromium/chromium/src/+/main:components/sync/engine/model_type_processor.h;l=18?q=modeltypeprocessor&sq=&ss=chromium

[31] https://www.chromium.org/developers/design-documents/sync/unified-sync-and-storage-overview/

[32] https://source.chromium.org/chromium/chromium/src/+/main:components/sync/base/model_type.h;drc=6fa3b54df432518a2966cfab92bed34ba95c6363;l=0

- Auto fill wallet offer (offers and rewards from user account)
- Themes
- Typed urls (info typed in omnibox)
- extensions
- search engines (for custom search engines)
- sessions (object for open browser session, used for tabs and history)
- apps
- app settings
- extension settings
- history delete directives (settings as to when history has to be deleted)
- dictionary (custom spelling dictionary entries)
- reading list (iOs only)
- user events (commit only user events)
- nigori (encrypted keys that are used to encrypt and decrypt the sync data) [33]
- user consents (commit only user consents)
- send tab to self (tabs sent between devices)
- security events (commit only security events)
- web apps (web app object)
- wifi configurations (wifi network configuration + credentials)
- sharing message (commit only sharing message object)
- workspace desk (webAuthn credentials)

These fields tie back to the data types that can be chosen by the user to sync, though some are mandatory, like nigori.

[33] https://source.chromium.org/chromium/chromium/src/+/main:components/sync/protocol/nigori_specifics.proto

META-DATA

- device info (client specific meta-data)

  This includes a cache guid as to identify a sync client on the current device; device info, like model of the device and device manufacturer; last updated, device type (platform of device); sync user agent; last updated timestamp; signin scoped device id, a device id that is stable until the user logs out.

- priority preferences (meta-data about what data types gets synced first, never encrypted)

- supervised user settings (never encrypted)

  This includes the settings: if sync is turned on, which data types should be synced, if an initial sync setup has been completed.

DATA TYPES

Each entity data structure [34] that is used for Chromium Sync for almost all syncable data types has the following data structure:

- id (server assigned Sync Id, or temporary a client assigned Sync Id)
- client tag hash (hash based on client tag and model type, used for map lookups)
- server defined unique tag (to identify this item as being a uniquely instanced item)
- name (for debugging)
- specifics (model type specific sync data (encrypted))
- creation time
- modification time
- is deleted

CONCLUSION

The data types and meta-data are pretty similar to Firefox Sync, see chapter 4. The unencrypted meta-data is mainly concerned with tracking sync data and client information that is not connected to a real person.

The meta-data that is stored on the Google server is not identifiable for a specific user unless authentication has been breached. This is because the syncId is a hashed field. The most important assets are the data itself, which is encrypted, and the keys to decrypt this data are shared between clients via the server but require the users pass phrase.

The meta-data that is shared is protected with authentication and HTTPS. The device info meta-data is encrypted as well so that Google servers do not get some information about the devices used, like device model and manufacturer information, which are, however small, indications that could help with identifying a user.

[34] https://source.chromium.org/chromium/chromium/src/+/main:components/sync/protocol/entity_data.h

8.4. Observed Implemented Privacy design patterns in Chromium Sync

For the context-relevant privacy design patterns, not much has changed in comparison with Firefox because of the basic use case that Chromium Sync also tries to achieve. This leaves the same list as for Firefox, see chapter 4.

The same patterns are also addressed in the Firefox chapter 4, and most of these same points stand in the discussion of Chromium Sync.

Onion routing

Onion routing has not been implemented. Chromium communicates directly with the Google servers. The lack of implementation of this pattern is not a relevant privacy risk because authentication is needed via a pass phrase or a Google account password, which via authentication enables HTTPS to the Google servers. Because of HTTPS, in theory, the connection is secure and no additional protocols are needed to hide sender and destination information purely for the meta-data.

Use of dummies

Use of dummies has not been implemented, as there is no evidence of the Chromium Sync Client manufacturing fake sync actions. The non-implementation of use of dummies does not pose a great privacy risk. In the case of a HTTPS breach it could be useful to have dummy actions, as to confuse a possible attacker that monitors the connection between client and server.

Anonymity set

For Chromium, an interpretation of this pattern would mean the clustering of related data that is now separate, like for instance, the auto fill data. A mitigation like this would mean that less meta-data is gathered and is traded for more processing power at the client-side to unpack the related data and inspect changes. In this case Firefox is different from Chromium, seeing as it does not subdivide data types like auto fill information.

In theory, all sync related information for a client could be packed into one blob of data that is then unpacked at the client-side and inspected. How far this proposal would have to be implemented and what risk severity it would avoid is out of scope for this study.

Encryption with user-managed keys

Encryption with user-managed keys has been implemented. Chromium encrypts the browser data that is to be synced on the client-side and then sends it to the Google servers. To encrypt and decrypt, the password or pass phrase of the user is used to generate a master key. In combination with HTTPS this lowers the risk of the data leaking to an adversary.

There is more risk in Chromium than in Firefox because the server side is closed source and stores the data and meta-data. Based on authentication it could in theory try to track which client information belongs to which data id's, and build a profile in this way. It is however not clear if this is even possible, but the link between client info and meta-data and the data itself has to be there somehow because the server has to know what data to sent to which client. The meta-data and its usage is probably not at great risk of being coupled to a specific user however.

8.5. Conclusion

The overall use case and architecture is similar to Firefox, except it uses the Google services for user authentication and Sync data distribution. Sync is fairly simple at a high-level and

is set up to not give Google all the information, by for instance, hashing the syncId that is used to identify the sync data for this user. Chromium Sync also has more options for the user for datatypes to sync and handles more meta-data and data sets. In this way more meta-data is collected that could have been minimized.

Though this is the case, the privacy design patterns that were possible were the same for Firefox as for Chromium because of the overall functionality: the basic distribution of data across devices. Thanks to having all the processing on the clientside except for storing data, similar to Firefox, encryption with user-managed keys is indeed implemented. The other three privacy design patterns were not a great risk, similar to Firefox, but the risk for Chromium overall is greater thanks to the use of Google services and less minimization of data and meta-data. This risk part is therefore less related to the specific implementation but more to the data and its handling.

9. Method Reflection

A lot of emphasis of this study was placed on the method and exploring the capability of privacy design patterns as a way to study a software system with regards to privacy. This sub chapter will address how to make the method more general (RQ4), observations around using the method and what was viable with regard to privacy design patterns and this method to study a software system.

The privacy design patterns are very general and address broad problems and solutions with regards to privacy. This means that some of them can almost always be found in a random software system that uses data that could be handled in a privacy unfriendly way. Because of their broad nature, it has been found that the relevancy of patterns in a system can be filtered out with general information about the architecture, large components and the data, meta-data and its structure in the software system. The patterns themselves address use case that transcend a client-server setting, and were set up in a general way, which suggests in combination with these findings that using privacy design patterns in other systems in general is a viable tool to study a software system.

The filtering of privacy design patterns based on relevancy in a software system is one thing, but finding the implementation is another. Conclusions around this search can also be made with a lot of general information about the system, in part because of the nature of the privacy design patterns themselves, but the overall picture can lead to a lot of details that determine whether a pattern has been implemented correctly or sufficiently as to adhere to the solution of the privacy design pattern, and not be a viable privacy threat anymore. With this manual method it has been found that a discussion can be raised surrounding the implementation of the patterns, and they can be identified based on large components and information surrounding the data, but an exhaustive detailed discussion is out of scope because the manual approach is simply too time consuming as of now.

The overall method of this study was pretty simple and can be copied in other systems because it is not exclusive to client-server systems. The privacy design patterns can raise a good discussion surrounding privacy risk and problems/solutions, but solid conclusions with detail about the correct implementation or the lack of a privacy design pattern will likely need an automated approach for this to be viable in even this small of a use case.

In the functionality of Firefox and Chromium Sync the exact implementation and details in the larger components of Sync were not as interesting, but only with regards to the data and if it is modified in some way. For a larger system it would also be efficient to start at a higher-level view of the system and determine which parts of the software only have to be checked for how it handles the data, seeing as the privacy design patterns in the Hide, Abstract, Separate and Minimize category are mainly concerned with data handling.

The data flow diagram approach is not a bad approach because it sticks to high-level components and a focus on the data itself. The privacy design patterns seem to tread the same space and can be filtered out based on these basic findings about the system. The privacy design patterns can also give a sense of focus for a layperson with regards to the system in study, as in, not a developer of the system in study.

10. FUTURE WORK

For future work the first thing to a dress is the complexity of most software systems, the details for privacy and what determines good privacy implementation, and the general nature of the privacy design patterns. This leads to the first problem of this study, and any effort to study the privacy design patterns in practice, which is that to resolve all these complexities and generalities a lot of knowledge about the system is needed. For a software engineer that helped to built the system in question the privacy design patterns can be really practical from the start to raise a discussion about if the problem that the pattern discusses exists and how the solution can be reached for privacy with this privacy design pattern. For a layperson regarding the same software system this is not as obvious. To mitigate this a more automated approach is needed to study the system, with the goal to remove a lot of the complexity of the system and to focus on the main points that are needed to contextualize the privacy design patterns in this system.

The privacy design patterns that are relevant for this system could in most cases be filtered with fairly general information about a system, like for instance, the kind of data it stores, its architecture and how the data is handled and mutates throughout the system. This could in future work be implemented in a more practical way as to speed up the part of the study which is concerned with gaining knowledge about the system. A possible avenue to explore is to track the system with regards to the data and where this data is inserted into the system, where it leaves the system and where and how it is mutated and handled along the way. This would in theory give a road map to work from that could be pretty solid, seeing as the privacy design patterns are solely concerned with the discussion surrounding the data, not the concrete implementation or non-functional requirements like for instance the efficiency. The result of speeding up the knowledge gaining phase of a study surrounding privacy design patterns would be to be able to arrive more quickly in the exploration of the complexity of privacy in the specific context of the privacy design pattern that is being studied and the software system itself.

In these kind of suggestions emphasis should be placed on automatically increasing the modeling of the system and/or data in the system, not on automatically detecting the privacy design patterns in a system. This is because the privacy design patterns can be both very general and very specific regarding the context, problem and solution. Therefore human analysis (preferably by a privacy expert) is always needed to some extent, and the goal is to make his/her job more easy with regards to gaining knowledge about the system, which leaves more room to dive into the complexity of privacy, as stated earlier.

11. Conclusion

For this study two open source software systems (Firefox and Chromium Sync) have been studied to identify a set of privacy design patterns. For this purposes a method was set up, which was also explored for its usefulness. The privacy design patterns can say something about the privacy risks/ threats a system once the possible privacy design patterns in this scenario are identified and the implemented privacy design patterns are found.

In the study of Firefox and Chromium Sync it was found that four privacy design patterns were possible in this use case, even in the narrow scope of this Sync functionality, but only one pattern was implemented. These patterns were onion routing, use of dummies, anonymity set and encryption with user-managed keys. The latter was found in both Firefox and Chromium Sync, which in this narrow scope was a great boost to privacy, while the others were possible in theory, but would not add much to diminishing the risks this functionality faces.

It was found that the comparison of Firefox and Chromium took place with the encryption with user-managed keys pattern. The conclusion here is that there is more risk in Chromium due to the implementation of this pattern, which distributes more meta-data to the server and does not implement its own server, which carries more risk than in Firefox Sync.

The filtering of possible patterns from a greater list of patterns was viable using the high-level building blocks of the system, like architecture and the overall data structure. A Data flow diagram could be useful for this task, seeing as the privacy design patterns focus on the data itself and are pretty general, which in this case was found to be pretty effective for filtering the privacy design patterns that were possible in this use case. Identifying the privacy design patterns that were possible in Firefox and Chromium Sync was possible with both the documentation and source code, though this was a very time-intensive process that is of yet not efficient with a manual method.

For future work the method must be automated somewhat to speed up the information gathering stage of the system in study, which would allow a more detailed analysis of a software system using the privacy design patterns. This is crucial seeing the nature of the patterns, which are very general but still specific, and would facilitate a more detailed discussion surrounding privacy risks for the data and how a proposed solution like a privacy design pattern could mitigate the possible risks the data faces.

12. APPENDIX

These are all privacy design patterns that were investigated, which can be found on privacypatterns.org. They have been categorized in the differnet privacy strategies, which state their overall goal. The same categories are used at this website.

Privacy strategy: Minimize

- Strip inivisble meta-data
- Added-noise measurement obfuscation

Privacy strategy: Separate

- Personal data store
- User data confiement pattern
- Anonymous reputation-based blacklisting

Privacy strategy: Hide

- Encryption with user-managed keys
- Use of dummies
- Psuedonymous messaging
- Onion routing
- Psuedonymous Identity
- Aggregation Gateway
- Trustworthy privacy plug-in
- Anonymity set

Privacy strategy: Abstract

- Location granularity

The strategy categories that were used exclude the strategies Inform, Enforcce and Control because of the scope of this study and because these strategies are not as oriënted on the back-end of a system but also on policies for instance. In this study the scope was the back-end because in this way the source code can be used to assess the privacy design patterns.

Patterns that were not investigated within the chosen Privacy Strategies are Attribute-based credentials (404 on site) and Protection against tracking, because cookies are out of scope and is not back-end related.

BIBLIOGRAPHY

Bernhard J Berger, Michaela Bunke, and Karsten Sohr. An android security case study with bauhaus. In *2011 18th Working Conference on Reverse Engineering*, pages 179–183. IEEE, 2011. 9

Ted J Biggerstaff. Design recovery for maintenance and reuse. *Computer*, 22(7):36–49, 1989. 9

Ann Cavoukian et al. Privacy by design: The 7 foundational principles. *Information and privacy commissioner of Ontario, Canada*, 5, 2009. 8

Michael Colesky, Jaap-Henk Hoepman, and Christiaan Hillen. A critical analysis of privacy design strategies. In *2016 IEEE Security and Privacy Workshops (SPW)*, pages 33–40. IEEE, 2016. 8

George Danezis, Josep Domingo-Ferrer, Marit Hansen, Jaap-Henk Hoepman, Daniel Le Metayer, Rodica Tirtea, and Stefan Schiffner. Privacy and data protection by design-from policy to engineering. *arXiv preprint arXiv:1501.03726*, 2015. 8

Nick Doty and Mohit Gupta. Privacy design patterns and anti-patterns patterns misapplied and unintended consequences. 2013. 8

Organization for Economic Co-operation and Development. Oecd guidelines on the protection of privacy and transborder flows of personal data, 1980. 8

Erich Gamma. *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995. 8

Seda Gürses and Jose M del Alamo. Privacy engineering: Shaping an emerging field of research and practice. *IEEE Security & Privacy*, 14(2):40–46, 2016. 8

Jaap-Henk Hoepman. Privacy design strategies. In *IFIP International Information Security Conference*, pages 446–459. Springer, 2014. 8

Miguel Ehécatl Morales-Trujillo, Erick Orlando Matla-Cruz, Gabriel Alberto García-Mireles, and Mario Piattini. Privacy by design in software engineering: a systematic mapping study. *Avances en Ingenieria de Software a Nivel Iberoamericano, CIbSE*, pages 107–120, 2018. 7

Daniel J Solove. Conceptualizing privacy. *Calif. L. Rev.*, 90:1087, 2002. 7

Leo Van Audenhove, Anastasia Constantelou, Martijn Poel, Marc van Lieshout, Linda Kool, Bas van Schoonhoven, and Marjan de Jonge. Privacy by design: an alternative to existing practice in safeguarding privacy. *info*, 2011. 4, 8

Kim Wuyts. Privacy threats in software architectures. 2015. 9