Towards User Privacy for Subscription Based Services

Master Thesis

Author(s):
Benelli, Allan

Publication date:
2022

Permanent link:
https://doi.org/10.3929/ethz-b-000551577

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

SCS

Towards User Privacy for Subscription Based Services

Master Thesis

Allan Benelli

1st April, 2022

Advisors:
Moritz Schneider, Prof. Dr. Srdjan Čapkun
Department of Computer Science, ETH Zürich
and
Sabine Proll, Alain Brenzikofer
Supercomputing Systems AG, Zürich

Abstract

Today online service providers collect a massive amount of data. However, not all of this data is necessary for a service to work, for example our usage data. In this thesis, we explore an approach to obfuscate the actual usage of a subscription-based online service while maintaining the possibility of restricting access to paying customers.

We introduce a system called MIXNET that allows users to use a service while obfuscating their actual usage. To do so, we use the concepts of mix networks and K-Anonymity. MIXNET aims against an adversary that inspects HTTP requests addressed to it to identify users. Before requests reach the service, MIXNET generalizes them and mixes the subscription-related session cookies used by different users. We explore how generalization and mixing session cookies can be implemented using Trusted Execution Environments (TEEs). By doing so, we prevent session leakage or the disclosure of session-related personal data.

We present a full implementation of MIXNET using Intel SGX and demonstrate its use in three real-world applications: NZZ and Tagesanzeiger, two Swiss online newspapers, and the streaming service Zattoo.

Contents

Chapter 1

Introduction

By using the Internet, we generate many a lot of data. Different actors collect this data. These often include the service we use itself, for example, to improve its service as well as third parties such as advertising companies. If a service wants to collect data about us, it can very quickly do so by using web tracking technologies [62, 48]. However, collecting usage data can negatively affect users, depending on how a service uses the data, and raise privacy concerns. For this reason, awareness is increasing for privacy-enhancing technologies and a lot of work has been done in this area. Examples are the Tor network, which is used for anonymous web browsing, privacy-enhancing proxies like Privoxy [22], or applications like the browser extension TrackMeNot [37, 31], which use the principle of obfuscation.

Despite considerable work in this area, there are some pitfalls. On the one hand, these measures do not provide users with absolute protection and can not obfuscate all of a user's usage. If a user wants to use subscription-based services, he must log in with his account. Consequently, all his requests are bound to his account once logged in. This causes data collection for a service that does not follow the principle of privacy-by-design. On the other hand, a user must be able to trust the privacy-enhancing technology provider not to work against him.

In this thesis, we show that by using Trusted Execution Environments (TEEs), we can obfuscate the connection between the request itself and the user who made the request. Our approach uses the concepts of mix networks and k-anonymity and applies two steps for each user request:

- Mixing user identities
- Generalizing the request

By using TEEs, on the one hand, we can perform these steps in a secure environment; on the other hand, a user can make sure that the expected code

is executed. This shifts trust from the privacy-enhancing technology provider to the TEE manufacturer.

To demonstrate this, we design Mixnet, a system that allows users to use third-party services while enhancing their privacy. The design of Mixnet maintains the service's ability to restrict access to paying customers. We implemented Mixnet entirely within Intel SGX [44, 17] and applied it to different real-world applications, such as newspapers or streaming services. Mixnet ensures that neither sessions nor session-related personal data of a user will be leaked. The principle of Mixnet does not require cooperation with the service provider. At the same time, a service provider could use Mixnet to improve its users' privacy.

Outline. This thesis is structured as follows: In Chapter 2, we introduce the problem, the adversary model, and the solution requirements. In Chapter 3, we provide the necessary background for this work. Chapter 4 gives an overview of Mixnet and addresses the requirements, followed by implementation details in Chapter 5. In Chapter 6, we present our results from applying Mixnet to real-world services and analyze its performance. In Chapter 7, we discuss the limitations of the selected approach and point out directions for future work. Finally, we finish with related work in Chapter 8 and conclude in Chapter 9.

Chapter 2

Motivation and Problem Statement

The Internet has become our daily companion, and when using it, we are continuously generating data. The sheer amount of generated and collected data is almost unimaginable and growing fast. Facebook generated 4 petabytes (PB) of data daily in 2014 [8] with 1.39 billion users [7], which corresponds to 2.87 megabytes (MB) daily per user. For 2020, IBM estimated that every person on earth generates 1.7 MB of data per second [19]. This data has a variety of origins: the use of web services and IoT devices, creation of social media content, digital photos and videos, messages, location data, or copies of this data on different servers.

However, not all user data is mandatory for the functionality of a service. Different actors have different motivations, collecting data about users' online behavior and creating user profiles. These include:

- Facebook for targeted ads [65]
- Netflix for their recommendation algorithm [38]
- Generally web services in the context of web analytics
- Intelligence agencies for solving crimes [50]

For a service, it is relatively easy to collect data about its users by using web tracking methods [62]. Common methods for web tracking include logging user behavior, combining and analyzing web traffic, tracking cookies, or other, more advanced, web tracking methods. There has been much research done in this area; for an overview, see [48].

Depending on how a service uses the collected data, it can have positive and negative effects [51] for the user. Below are some often critisized user privacy issues of current internet services:

- A service that is completely tailored to us and only shows us content that we like creates a "filter bubble", as Pariser calls it [55]. In

exaggerated terms, one could speak of a kind of "censorship" when platforms take away our decision on what content we should consume and withhold other content from us.

- If a service is compromised and personal data falls into the hands of unauthorized third parties.

- Potential sale or sharing of personal data. Many users are not even aware of what data a service collects about them or what the service does with it, as they often do not read the terms of use carefully.

- Much of what we do, write, or say ends up in permanent digital files and can eventually come back to haunt us. It is not easy to delete something from the web again, which lead to the saying "The Internet does not forget" [53]. This is an increasingly important issue, which is underscored by the European Union's General Data Protection Regulation (GDPR) [9], especially paragraph 66, which deals with the "right to be forgotten".

These privacy constraints have led to a wide array of research into privacy enhancing technologies. A well-known example of privacy-enhancing technology is Tor [30], used for anonymous web browsing. Tor conceals a user's location and usage from Internet service providers conducting traffic analysis or network surveillance. Another technology is Privoxy [22], a privacy-enhancing proxy that modifies web page data and HTTP headers and filters out malicious requests. Several browser extensions exist which enhance privacy. They block tracking cookies, filter requests, control and disable javascript on web pages or obfuscate usage by generating noise, like TrackMeNot [31, 37]. However, in the case of subscription-based services, these technologies do not provide complete protection, as a user has to log in with his account to use the service. Consequently, all his requests are bound to his account once logged in.

In this thesis, we focus on subscription-based services with a paywall, such as newspapers, streaming services, online software, or databases for images or similar. For a newspaper, for example, it is legitimate to restrict access to paying customers. However, for the functionality of the newspaper, it is not essential to know exactly which user has read which article. This connection, from users to articles, is what we aim to obfuscate in this thesis. Unlike services that are free to use, we claim that in an ideal world, when we pay for a service, the service:

- Should not make additional profit from user-related data.

- User-related data should be protected.

2.1 Adversary Model

In this thesis we focus on the following types of attacks. We assume an active network attacker. This attacker can try to perform various attacks such as man-in-the-middle (MITM) [49], DROWN [39], or Sweet32 [26, 42] attacks, to eavesdrop on communications. However, it cannot break the current confidentiality and integrity guarantees of modern transport layer security (TLS) [64, 59].

The service itself is not actively malicious. However, the service can collect logs of our requests and try to use them later on to collect user data. It can identify the sender of a request in two ways:

- either referring to the user's session cookie that was used to request access to the service.
- or combining multiple requests and mapping them to one user. To do so, it refers to the same headers, analyzes the clickstream, or combines content-related requests

We limit the service ability to use these methods of identification. It cannot use any other tracking methods, such as fingerprinting or tracking cookies.

We also consider malicious users who have the following intentions:

- try to leak personal information of other users
- try to obtain session-relevant information of another user to hijack his session
- try to circumvent the paywall

Finally, we consider an SGX attacker that tries to modify the code within an Intel SGX enclave or read out memory. He aims to capture sessions or session-related personal data to use it for his purposes. However, the attacker has no ability to break the security properties of a secure SGX enclave.

2.2 Requirements

In this thesis, we assume that a solution that enhances users' privacy by protecting users against such an adversary model must meet the following requirements:

- **Confidentiality and integrity:** The current confidentiality and integrity guarantees should remain the same.
- **No leakage of personal data:** Web services keep personal data about users; this data must be protected.
- **No session leakage:** It should not be possible to hijack a user session.

- **Restricted access:** In order to not violate the terms of use of a service and offer paid content to unauthorized users, access to a service must be restricted to paying users.
- **Trust:** Another system brings another level of trust. A user must be able to trust this system not to collaborate with the adversary or a third party, which would be even worse than the adversary's original data collection.

Chapter 3

Background

3.1 Web Authentication

Web authentication [47] is the process of verifying a user's identity before granting access to a web resource. It is a critical security measure to protect sensitive data and resources. Web users encounter many forms of authentication in everyday life, such as passwords, PINs, or biometric scans.

What happens after our identity has been verified successfully? All authentication mechanisms can be classified into either token-based or session-based authentication.

In *token-based authentication*, the server generates a token containing all the necessary user data and signs it with a secret key. The token is sent back to the user and stored on the user-side, but not on the server-side. The user typically sends the token in the "Authorization" header of an HTTP request to a server, which decrypts the token, verifies the signature, and grants access. This method is also often used for server-to-server connections.

In *session-based authentication*, the server generates a session, or session file, which contains all the necessary information about a user. This session is stored on the server-side, unlike the token-based method. The session ID, a randomized string to prevent third parties from extracting any information about the user, is server-side encrypted and transmitted to the user as a cookie [40]. These cookies should be encrypted to prevent any client-side tampering. In this thesis, we call this cookie the session cookie. The user submits this session cookie with each request to a server, which checks whether the session ID is present in the session store and then grants access. This method is used for user-to-server connections.

In this thesis we will almost exclusively deal with session-based authentication since the services we investigate rely on it. Anyone in possession of a session cookie has almost unrestricted access to the account associated with

it. For this reason, also client-side protection mechanisms are defined for cookies [40]. These include cookie-specific flags such as the `secure` attribute [40], which forces cookies to be sent only over encrypted connections, and the `httpOnly` [18, 40] attribute, which prevents cookies from being accessed by scripts.

3.2 Trusted Execution Environments and Intel SGX

A Trusted Execution Environment (TEE) is a secure area of a computer's main processor where applications can be run in a protected environment, isolated from the rest of the systems. TEEs are used in various applications, including mobile devices, smart cards, automotive systems, and on servers in data centers.

Intel Software Guard Extensions (SGX) [44, 17] is a modern TEE environment providing various protection mechanisms such as isolated code execution and attestation[2]. Intel introduced SGX as an instruction set architecture extension, available in Intel's processors starting with the sixth-generation Core processors (Skylake) in 2015. SGX allows developers to create multiple protected environments, called "enclaves". An enclave is a secure area of memory that is isolated from the rest of the computer's memory. Data and code run in an enclave is protected from being accessed or tampered with by other applications or the operating system.

Besides isolated code execution and attestation, Intel SGX offers many other protections, such as sealing and memory encryption. In the following, we give a rough overview of the relevant mechanisms for this work. For more detailed coverage, see [44].

- **Attestation:** SGX can prove that an enclave can be trusted by checking whether an enclave contains the expected code. This process is called attestation. Intel distinguishes between local and remote attestation:
  - **Local Attestation:** Two enclaves on the same platform can check each other.
  - **Remote Attestation:** Intel's online attestation service [3] is used to verify an enclave's signature. A system service called Quoting Enclave creates this signature. The verifier and the enclave to be verified do not have to be on the same platform.
- **Runtime Isolation:** As indicated, SGX provides isolated code execution. Through protections enforced in the processor, SGX prevents other potentially malicious software, processes, or the operating system from accessing code and data in the enclave.

- **Sealing and Memory Encryption:** SGX encrypts any runtime memory in an enclave, preventing an untrusted operating system from accessing it. In addition, SGX can protect confidential data used across multiple enclave executions. This process is called sealing and involves encrypting and authenticating enclave data.

3.3 Integritee

Integritee [16, 14] is a framework that aims to improve privacy by integrating TEEs into substrate-based blockchains [11]. Integritee provides users with a way to build decentralized, scalable applications that rely on multiple, mutually trusting TEEs. Intel SGX is used in Integritee, and the component containing the enclave-code is called a worker. Workers connect to nodes in the Integritee chain, which verify and store the attestation of the workers.

Integritee offers some key features that are relevant for this work:

- **Remote Attestation**: Integritee simplifies the process of remote attestation for examiners by storing the TEE attestations of all workers in an Integritee blockchain. Each worker transmits remote attestations daily. As a result, a verifier no longer needs to verify the signature of an enclave via a manufacturer's online attestation services.
- **Redundancy and Scalability**: Integritee is designed to expand the network with new workers without significant hurdles. A new worker can join the existing network by requesting a current worker to perform a mutual remote attestation. If successful, the new worker receives the state and the corresponding keys and immediately starts to work in parallel. An Integritee chain can attestate more than 1000 such workers.

For more in-depth coverage, see [34].

3.4 K-Anonymity

K-anonymity is a concept for anonymizing datasets, first introduced by Latanya Sweeney [63, 61]. The concept was introduced to address a problem: anonymized data can be re-identified by linking it with other datasets. Sweeney introduced the privacy model for use with databases containing personally-identifying information.

The concept of k-anonymity is based on the idea that any subset of records in a database will have at least k individuals who share the same identity attributes. Then a single individual cannot be identified using only those records, as we cannot distinguish which of the k individuals it is.

To achieve k-anonymity, each record in a dataset should be generalized and then possibly suppressed, according to its degree of risk for identifying an individual:

- **Generalization**: Generalize the values in this category. The following would be examples: replace age with age groups, cities with countries, or religion with a boolean value for ”believing”.
- **Suppression**: Remove or replace a value with an ”*”. This is mainly done when a category has less than k values.

Chapter 4

Overview

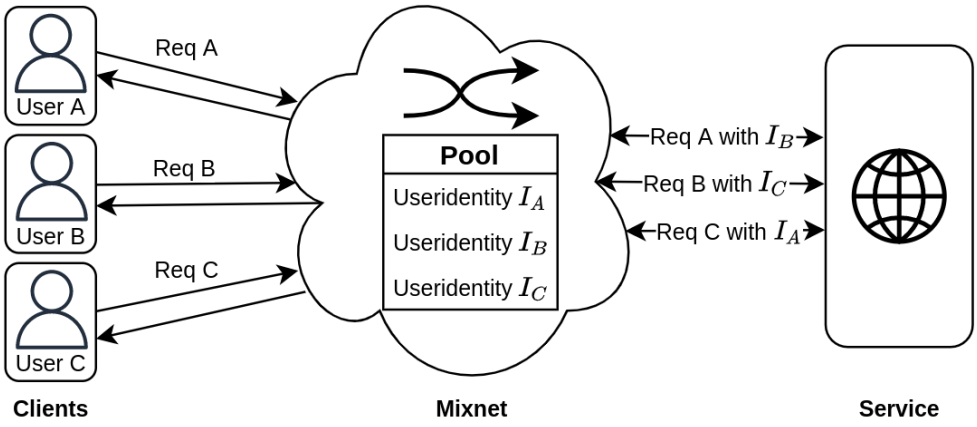

Figure 4.1: Overview of MIXNET

MIXNET is a service, completely running in a Trusted Execution Environment (TEE). It allows users to use third-party services while enhancing user privacy. As shown in Figure 4.1, the main idea of MIXNET is to mix user sessions with the goal that a service can no longer identify individual users. Accordingly, MIXNET helps to prevent data collection.

Instead of using the service directly, users will access the service through MIXNET, analogous to a proxy service. MIXNET keeps a pool of user identities using its service to perform its primary task of forwarding user requests with the identity of another user. In doing so, MIXNET applies the concept of k-anonymity to HTTP traffic from different users by generalizing or removing fields from HTTP requests.

This way, we enhance privacy, as the service can no longer assign the traffic

to a specific user; it only knows that the request came from a user in the group of Mixnet users. With this solution, we do not harm a service by disturbing its traffic or falsifying any statistics about page views or the like. This is important because we assume services that neither work with us nor against us. Therefore, it would be counterproductive to harm such a service.

Mixnet fulfills the requirements from Chapter 2 as follows:

- **Confidentiality and integrity:** Mixnet represents a man-in-the-middle [49], as it sits on the channel between a user and a service. To secure this channel, the entire traffic between clients and Mixnet and between Mixnet and services uses the Transport Layer Security (TLS) protocol [64, 59]. As a man-in-the-middle, Mixnet can read the messages in plaintext, but since it runs in an SGX enclave, this data cannot leak to an adversary. Therefore, Mixnet does not degrade the security of any message between the user and the service.
- **No leakage of personal data:** Mixnet ensures that users' personal data is not leaked to another user using two approaches. First, Mixnet blocks requests to user-account-specific pages, as these mainly contain sensitive data. Second, Mixnet detects and replaces personal data embedded in other pages.
- **No session leakage:** Mixnet runs entirely within a Trusted Execution Environment (TEE), so we rely on the security properties of the TEE to protect active user sessions. Mixnet cleans up the service response and removes session cookies or corresponding references. This ensures that a user does not get any hint about the session used.
- **Restricted access:** Mixnet keeps track of which user submits which session cookie. These cookies are regularly checked for validity, and a user has access as long as his cookie is valid. This ensures that only authorized users have access to a service.
- **Trust:** Remote attestation, which TEEs provide, allows a user to ensure that the expected code is running. To simplify attestation, we rely on Integritee, and the code of Mixnet is open source [10].

Chapter 5

System

In this chapter we present the design of Mixnet. The selected approach supports multiple real-world services, which users can select by setting a cookie.

The main purpose of Mixnet, mixing the user identities, is a simple task. Recall that tokens or session cookies are used on the web to identify a user, and having either of them grant unrestricted access to a service. There are two ways to get such a cookie: Either the user gives it directly to Mixnet, or he provides his credentials, and Mixnet performs the login to the service, which then returns the cookie. The first way is chosen for Mixnet, as it is less complicated since it does not have to deal with complex authentication mechanisms like multi-factor authentication. It has the advantage that Mixnet does not have to worry about sensitive user credentials and only stores and protects session cookies.

5.1 Architecture

Mixnet [10] builds on the code of an Integritee-worker [15] that uses the Apache Teaclave SGX SDK [1], a software development kit for developing Intel SGX applications in Rust. Mixnet consist of three modules, as shown in Figure 5.1. All modules reside in a single Intel SGX enclave.

5.1.1 Server

The server module provides a multithreaded TLS server, handling multiple TLS sessions concurrently. To do so, it uses Mio [29], a fast, low-level Input / Output library, which notifies if there is either a new connection to accept or new network traffic over an existing connection. To protect and encrypt the traffic it uses the TLS library rustls [24]. Rustls performs a TLS Handshake to establish a new connection. It handles encryption and decryption using

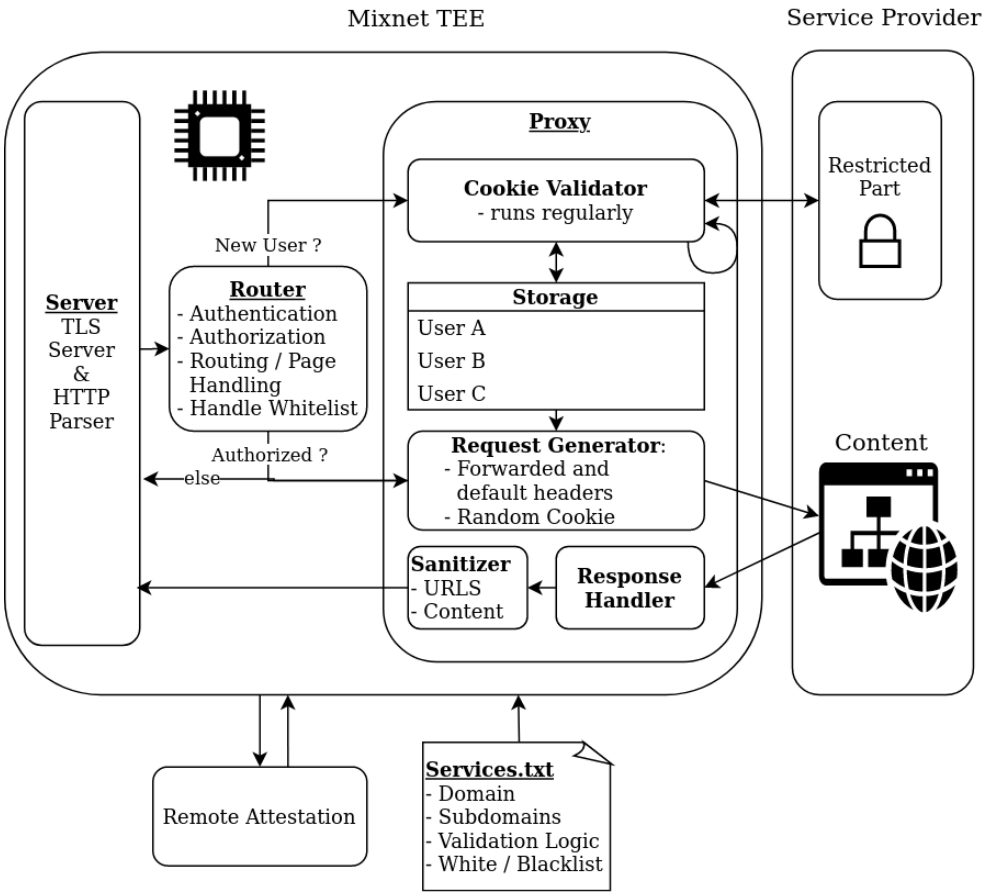

Figure 5.1: System overview

the server's certificate and private key. The server reads and decodes new data and parses it to a valid HTTP request. If parsing is successful, the server uses the request's headers and body to obtain the information the router module needs to process the request; otherwise, the server returns an error message to the client. Necessary information includes the UUID to uniquely identify the user, which service provider and resource he is targeting, and his session cookie for the service provider if it is the user's first request for this specific service provider. Establishing a new connection is handled by the main thread while handling events from an existing connection is passed to a thread pool.

5.1.2 Router

The router essentially hosts the frontend of Mixnet. This frontend allows new users to log in and select which service they want to access. The router either returns the pages of the frontend directly without querying any third-

party service, or prepares user-supplied data for the proxy to the third-party service.

Frontend

For users which are not yet authenticated to MIXNET, we first have to check if they request a valid path. To do so, we use a modified version of the library route-recognizer [13], which recognizes patterns in the requested path. The router returns the requested page of MIXNET for valid paths. Otherwise, the router returns a default error page and an HTTP 404-response. To log in, a user should request the main page of MIXNET, on which he can select the desired service provider and transmit his corresponding session cookie. The main page sets the selected service provider as a cookie for the user, which we call the target cookie, and assigns a UUID if one does not already exist. This UUID is later used to uniquely identify users on MIXNET.

For users already authenticated to MIXNET, the router checks if they are authorized to access the requested service provider. It does so by checking the following two conditions:

- Whether the cookie storage contains a valid session cookie for the respective service provider of this user.
- Or if the request contains a session cookie for the respective service provider. This session cookie is passed to and checked by the cookie validator from the Proxy module, described in subsection 5.1.3. Valid cookies are stored together with the UUID of the owning users in the cookie storage.

If one of these two conditions applies, the user is authorized to access this service provider.

Proxy-Preparation

To prevent personal data leakage, we must prevent users from requesting a path that would lead to a page showing personal data associated with the session cookie used for this forwarded request. The router module checks if a user is authorized to access a specific path by having multiple route-recognizers instances, one per service provider. The router builds the route-recognizer for a service provider using a predefined white or black list of which paths users may access and which not.

These route-recognizers, as well as the previously mentioned one of the frontend, support the following formats, among others:

- exact path segments of the form **/a/b**.
- named parameters of the format **/a/:b**, whereas "**:b**" captures the filename at location "**:**" in a parameter named "**b**".

- path wildcards of the format **/a/***.

Thus, for a specific service, we can either directly exclude individual path patterns that lead to content with personal data or, conversely, only allow individual path patterns that are necessary to use the service and block the rest. The first variant has the disadvantage that the requirement is no longer fulfilled if a service provider changes paths on his side. In contrast, the second, more restrictive variant continues to function. With both variants, it can still happen that personal data is included as part of the content, for example, in the navigation bar with a link to the user profile or as part of a personalized greeting. The sanitizer part in the proxy module, which is described in more detail in the chapter 5.1.3, handles these cases.

The router forwards authorized requests to the proxy module and returns a 403 error page in case of unauthorized requests.

5.1.3 Proxy

The proxy module itself consists of five parts, the cookie validator, the cookie storage, the request generator, the response handler, and the sanitizer:

The **cookie validator** verifies the session cookies that users submit and is thus a central element for access management. At the same time, the cookie validator extracts personal user data associated with the session cookie, which must not be disclosed when using this cookie. The sanitizer uses this data in a later step.

A flow must be defined for each service provider. Such a flow can include multiple HTTP requests with the cookie to be checked. The target of a request is an area of the web service restricted to authorized users or a specific API endpoint to query the subscriber status. The response body and its status code provide information about a user's access authorization. Based on the response, we can check if a session cookie is invalid, as shown in the following listing:

- Status code 302: Redirection to a login page means the session cookie is invalid.
- Status code 403: Missing authorization for the requested page means the session cookie is invalid.
- API request: JSON response contains a subscription field. If the value is equal to "false", the session cookie is invalid.

Otherwise, it was a valid session cookie, and the validator can extract, either from the same response, or using another request, the personal data for the sanitizer. This personal data includes the name and the mail address and further potential data like the residential address, an identity token, or a User ID.

The validator stores the session cookie and personal data, as well as the UUID of the user in the **cookie storage**. All these cookies in the storage are checked for validity by the validator at regular intervals. Invalid cookies and their associated data are deleted from the storage, and the user who contributed the cookie is denied access. This periodic validation ensures the following properties:

- The cookie storage contains only valid session cookies, which are required to use a service.
- There is no violation of the Terms of Use because only paying customers of a service provider have access.

The **request generator** is responsible for making the actual HTTP request to the service provider. To do this, it takes the request that the user initially made to MIXNET, extracts the path, and the target cookie, which defines to which service provider the request should be sent, any extra headers, and the body. The generator creates a valid target URL from the path and the target cookie. We defined which headers the request generator should take over from the original request and which are not used further. In addition to this, there is a set of default headers that the request generator adds to each request. The set of default headers include, for example, the "User-Agent"-Header, so that browser information is not unnecessarily disclosed to the service provider, and the "Content-Type" and "Accept" headers as some servers would otherwise reject the request. Additionally, a valid session cookie is added to access the service provider, randomly selected from the cookie store.

Once the generator has built the HTTP request, it checks if there is already an open TLS stream to the host of the target URL. If so, the request is sent over this stream. Otherwise, it opens a new secure connection to this host and sends the request over it. If the request fails, e.g., because of a broken stream, the request generator tries to open a new stream and resend the request. If the request is successful, the generator passes the response to the response handler.

The **response handler** decides what to do, depending on the class [12] of the response status code:

- **Informational responses (100-199):** The response's status and body are passed directly to the user.
- **Successful responses (200-299)**: The response body is passed to the sanitizer part. The sanitizer will solve two problems: On the one hand, the answer contains relative and absolute paths through which one would land on the original website. Therefore, the sanitizer must slightly edit the content so that the system continues to work. On the other hand, this response may contain user data that should not be

disclosed to other users, which the sanitizer will replace. Afterward, the cleaned-up body is passed with the original status code to the user.

- **Redirection messages (300-399)**: The "Location" header of the response is intercepted and adjusted so that the redirection routes the subsequent traffic through Mixnet. Otherwise, a user would be redirected directly to the service provider.
- **Client error responses (400-499)**: The server's corresponding error message and response are returned to the user.
- **Server error responses (500-599)**: A default response is returned to the user.

The functionality of the response handler regarding headers is similar to the request generator. The headers that the user receives consist of forwarded headers, default headers, and headers computed directly by the response handler. As a principle, the response headers returned by the server are not used except for the most necessary ones. Therefore tracking cookies or similar are not forwarded to the user. Exceptions include the following headers: `Cache-Control`, `Content-Type`, `Date`, and `server`, which are forwarded not to impair the functionality of the individual services. After completing its task, the response handler calculates the "Content-Length" and uses it to create a valid HTTP response, which is returned to the TLS server. The TLS server forwards this response to the user via the secure connection.

As mentioned before, the **sanitizer** solves two problems:

- Intercept absolute and relative paths of a website and, if necessary, rewrite them to ensure that a user does not land on the service's website and that future accesses continue to go through Mixnet.
- Prevent the disclosure of a user's sensitive data.

The sanitizer solves both problems using Regular Expressions, each with a different approach. In the path problem, we have to distinguish between the main and subdomains of a service:

- Relative paths of the main domain are easy to handle since the host and the path are transmitted separately in HTTP requests. Together with the target cookie transmitted, this corresponds to how our TLS server processes requests.
- Absolute paths of the main domain are intercepted using Regular Expressions. The domain in the absolute path is replaced with the domain of our system. The TLS server can handle this case the same way as the relative paths.
- Subdomains are even less demanding. Per service, subdomains can be defined, which are stored as Regular Expressions. The Regular

Expressions match URLs with these subdomains, replace them with this system's domain, and append the original URL as a query string. The TLS server checks for each request whether such a query string exists and sets the destination address accordingly.

In order to protect the sensitive data of the cookie owner, the sanitizer has to search the content for it. Recall that the cookie validator stores the sensitive data with the cookie in the cookie storage. Having the data already makes it easy to search for it with Regular Expressions and replace it.

5.2 Workflow

Figure 5.2 summarizes the system described in the last chapter and shows an overview of all the requests that happen, when a user uses the service:

(1) The user enters the domain of Mixnet in his browser. The browser and the TLS server in Mixnet perform a TLS handshake.

(2) Mixnet returns an HTML page where the user can select the service provider and enter his session cookie.

(3) The user selects the service provider he wants to use via Mixnet and enters his session cookie.

(4) To check the cookie, the cookie validator creates a new request using the request generator. If no secure connection to the service provider exists, the request generator opens a new one and performs a TLS handshake.

(5) The cookie validator verifies the cookie by sending a request to the service provider and inspecting the response.

(6) If the cookie from (5) was invalid, a 403 error page is returned to the user. The user can then start again with step (3).

(7) If the cookie from (5) was valid, the service's home page is requested via the request generator, using a random cookie from the cookie storage.

(8) The response is processed by the response handler and the sanitizer and returned to the user.

As of this step, the user has successfully established a session with the service via Mixnet. The user now sees a slightly modified version of the original service provider's home page and can navigate it.

Steps 9-11 represent all other requests routed through Mixnet:

(9) A request for a resource is sent by the user.

(10) Mixnet verifies that the user is authorized to send this request. The verification includes:

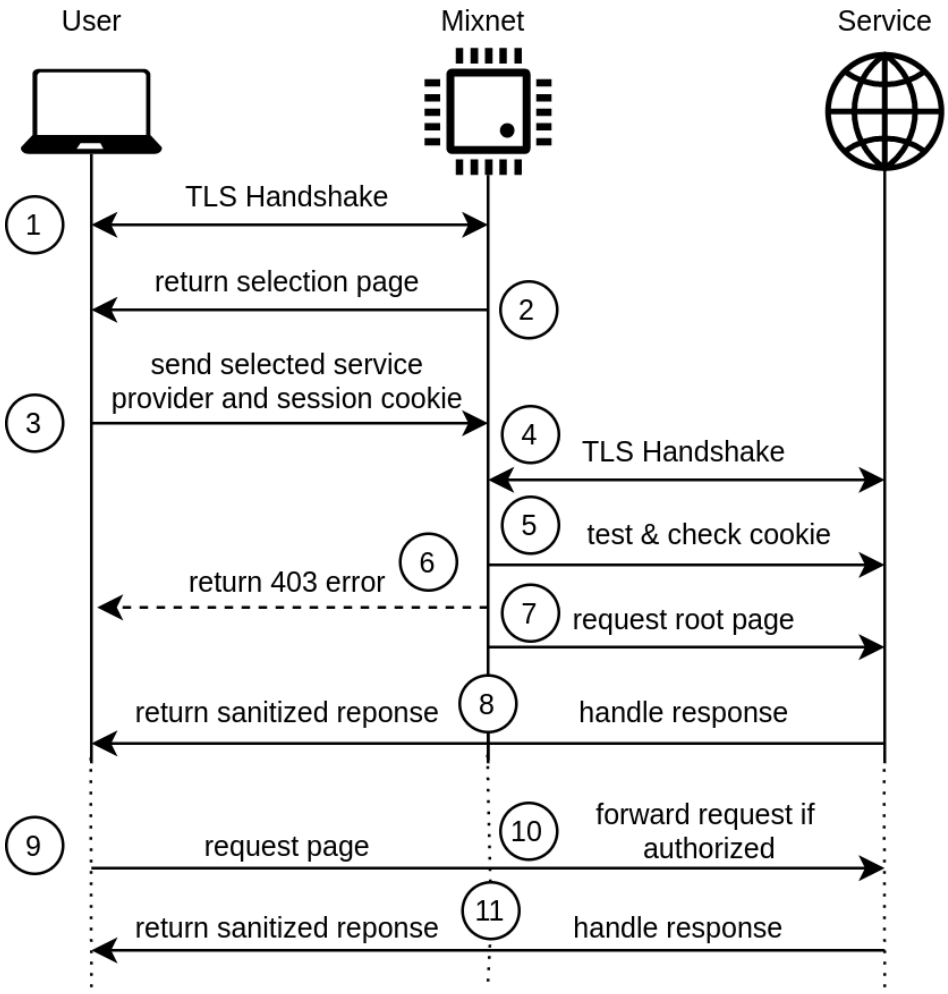

Figure 5.2: Workflow

- Valid cookie in cookie storage; otherwise, a new session cookie must be submitted, which in turn must be verified.
- The requested resource does not contradict the white/black list rules.

(11) same as step (8).

The user's session is valid until the cookie validator invalidates his initially submitted session cookie. A user can interrupt the session by deleting the target cookie. Recall that the target cookie defines which service the user has selected. The sanitizer adds a button in HTML responses to simplify deleting the target cookie. The user is then redirected back to Mixnet's selection page. He can either select another service and submit a session cookie or resume the existing session with the previous service without submitting a session cookie again.

Chapter 6

Evaluation

We have tested Mixnet on three real-world subscription-based services with a paywall. These include two Swiss daily newspapers, NZZ [20] and Tagesanzeiger [27], and the online streaming service Zattoo [35]. All three services use a different approach regarding web authentication and session management. Tagesanzeiger uses OneLog [21], a cross-platform login solution used by several Swiss media. NZZ currently uses its own solution but is expected to join OneLog in 2022, and Zattoo uses Beaker [4], a Python library for web sessions. The following challenges arose during the service-specific implementation:

- **Tagesanzeiger:** When a user navigates through the webpage by clicking links, Tagesanzeiger tries to revalidate a users' session-cookie using client-side Javascript. This revalidation will always fail, as Mixnet does not forward the used cookie to users. Mixnet handles this by triggering a full page reload when detecting such a failed revalidation.
- **NZZ:** NZZ has a Javascript-heavy page, dynamically creating the URL of assets like images while rendering on the client-side. To ensure that all the traffic is routed through Mixnet, we used Request Control [32], a Firefox browser extension, to redirect requests. We note that the same can be achieved either by

  (i) appending a custom javascript to the page that performs the path rewriting, or

  (ii) by rewriting the existing javascript to generate only URLs that point to Mixnet.

  However, due to the implementation complexity of these approaches we used the browser extension Request Control as a substitute.
- **Zattoo:** Due to the use of Beaker, a single session cookie is not enough to get full access. For Zattoo to work, a user must supply the Zattoo-

specific and Beaker-specific session cookies, which must always be forwarded together by Mixnet.

6.1 Performance

We measured Mixnet's throughput and the generated overhead using Mixnet to access the implemented services. Mixnet was run on an Intel i7-6600U machine with 16 GB RAM and connected over WLAN 802.11ac to the Internet and the local network for all measurements.

To determine Mixnet's throughput, we measured how many requests per second Mixnet can process and how many TLS handshakes Mixnet can perform per second. We consider TLS handshakes because they initially impact throughput since each new secure connection must first be established. We set up a second, much simplified TLS server on an Intel i7-11800H machine with 32 GB of RAM and also connected it over WLAN 802.11ac to the local network to perform our measurements. In the following, we call the second TLS server the native TLS server. The native TLS server was implemented on the same code base as the TLS server in Mixnet, but unlike the one in Mixnet, it does not run on a TEE and does not parse requests but always returns a minimal response. The overhead generated by the native TLS server is negligible compared to the measured latencies while using Mixnet, as can be seen from the measurements in A.1 in Appendix A.

	Mixnet	**Native TLS Server**
AVG	203	456
MIN	171	373
MAX	304	508

Table 6.1: Comparison of TLS handshakes per second

We used `tls-perf` [28] to measure the number of TLS handshakes per second. Table 6.1 shows the results of a 4-minute measurement for Mixnet and, in comparison, the results of the native TLS server. Mixnet can only handle about half as many handshakes as the native TLS server, with an average of 203 handshakes per second. This performance loss is mainly due to the isolated code execution and the associated overhead of calls between the operating system and the enclave.

To determine how many requests Mixnet can handle per second, we used a modified version of `plow` [25], which allows us to define the number of concurrent users and the number of requests per user. With `plow` we measured different scenarios by sending requests to the native TLS server over Mixnet. This way, we avoid possible network overhead and throttling

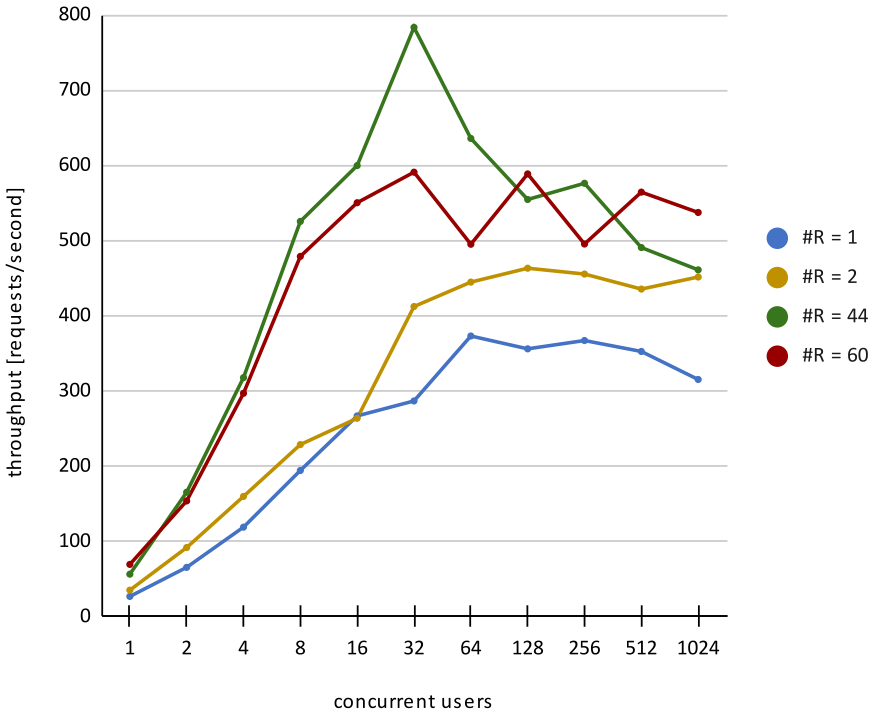

Figure 6.1: Throughput for different scenarios where #R := requests per user

due to possible protection mechanisms of a service provider. To get scenarios as realistic as possible, we set the number of requests per user according to the services we implemented in Mixnet. For Tagesanzeiger and NZZ, we took the number of requests that run through Mixnet when a user requests the home page of the specific service. This resulted in 44 requests per user for NZZ and 60 requests per user for Tagesanzeiger. For Zattoo, we have set the number of requests per user to 2, which corresponds to the number of average requests per second during actual streaming. Additionally, we created a scenario where each user sends only a single request for reference. We measured these scenarios with different numbers of concurrent users five times each. Figure 6.1 shows the average measured throughput. For scenarios that send many requests per user, the throughput settles between 500-600 requests per second. Scenarios that send few requests have lower throughput because TLS handshakes dominate the overhead.

We measured the overhead of Mixnet for a single user based on the implemented services. For Tagesanzeiger and NZZ, we measured when the DOMContentLoaded event was fired and when the page was completely loaded. Table 6.2 compares the mean times of 5 runs for accesses via Mixnet

Service	Access	Initial HTML (± std)	Full pageload (± std)
Tagesanzeiger	direct	254ms (± 41)	2.628s (± 0.239)
	Mixnet	458ms (± 89)	2.054s (± 0.774)
Overhead		**+ 204ms (80%)**	**- 0.574s (21%)**
NZZ	direct	289ms (± 35)	2.802s (± 0.261)
	Mixnet	304ms (± 74)	1.49s (± 0.243)
Overhead		**+ 15ms (5%)**	**- 1.312s (46%)**

Table 6.2: Mean times of DOMContentLoaded event (Initial HTML) and full pageload accessing a service directly or via Mixnet

and direct accesses to the service. The results show that Mixnet takes a little longer to load the initial HTML document but loads the entire page faster. The former is the overhead of accessing via Mixnet; the faster page load can be explained with the defined white-/blacklists in the router module: Mixnet blocks some requests and returns an early response. These are mostly requests to load advertisements or other content from third parties and are not necessary for the service to work. For Zattoo, we streamed the same TV-Channel directly and via Mixnet, and for video segments, we compared the round-trip times (RTT) of each request. Figure 6.2 shows a mapping of these requests, and Table 6.3 summarizes the results, showing that streaming over Mixnet only incurs about 50ms of additional latency per request. The three outliners seen in Figure 6.2 are due to network conditions and reestablishing TLS sessions with Zattoo's servers.

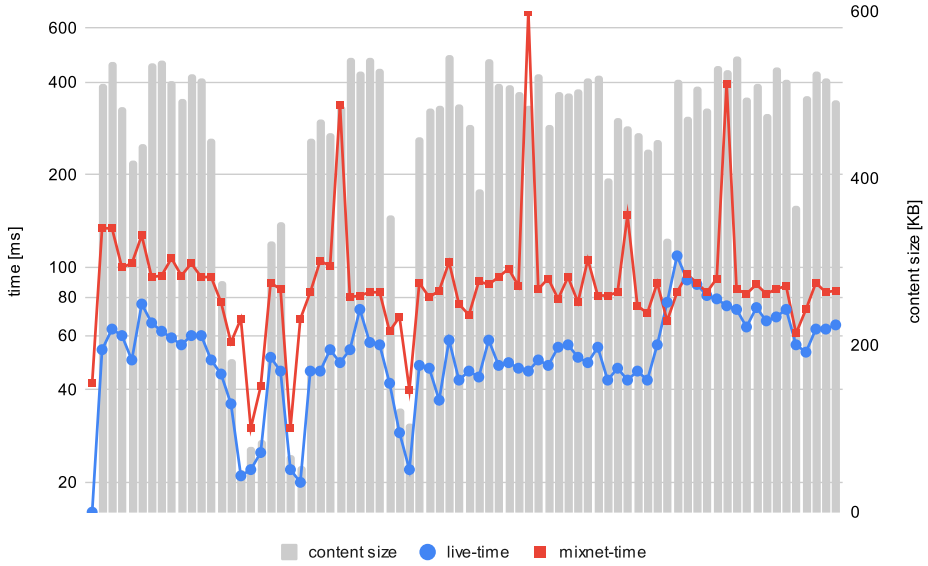

Figure 6.2: Round-trip times (RTT) of video packages while streaming Zattoo via MIXNET and directly.

	Direct	MIXNET	**Overhead**
Mean RTT	53ms	99ms	+46ms

Table 6.3: Mean round-trip times (RTT) of video packages while streaming Zattoo directly and via MIXNET.

Chapter 7

Discussion & Limitations

The principle of Mixnet works successfully for three real-world services, but it runs into limitations at a few points. We address and discuss these limitations in this chapter.

Adversary Model. We have limited the capabilities of our adversary to identify users based on HTTP requests. If we extend the adversary with, for example, the ability to use tracking cookies, then Mixnet partially reaches its limits. Cookies generated on the server-side and returned with the response can be handled by Mixnet alone since it does not forward response headers, and thus the generated cookies, to the user. However, embedded javascript could generate cookies on the client-side and send them directly to the adversary. This would circumvent the security mechanism provided by Mixnet. For such cases, additional mitigations are needed, such as the browser extension Request Control we used for NZZ, which would route the request through Mixnet. Mixnet would intercept the request and not forward the cookie to the adversary. Further countermeasures are also required if we extend the adversary with other capabilities, such as canvas fingerprinting. Canvas fingerprinting requires access to the HTML5 Canvas API as well as javascript. A simple way to prevent canvas fingerprinting would be to disable javascript. There are also various browser extensions to do so. For more tracking methods and mitigations, refer to [62, 48]. If we extend our adversary model to an actively malicious service provider, further measures are necessary. Such a service provider could actively work against Mixnet, for example, by blocking Mixnet's IP address. In such a case, new IP addresses would have to be constantly used for traffic originating from Mixnet, e.g., using different proxies or using Tor's concept of anonymous communication. In this case, the service provider would take further measures against Mixnet, ending up in a cat and mouse game. The use of further privacy-enhancing technologies would also be required in the case of a malicious SGX provider on which Mixnet runs. Due to the use of

TEEs, the malicious SGX provider cannot inspect the requests themselves or access any data in the enclave. However, a malicious SGX provider would learn the IP addresses of the user. Together with the service provider, he could thus reveal the identity of a user. To prevent this, the IP address of the original user would have to be obfuscated. Based on all these examples, we strongly recommend using Mixnet side-by-side with other countermeasures to protect privacy effectively.

Vulnerabilities. We rely heavily on Intel SGX's security guarantees, although there are known vulnerabilities in Intel SGX. For more in-depth coverage of known vulnerabilities and mitigations, see here [58, 45].

Personal Data. Filtering out a user's personal data requires a lot of manual effort. When setting up Mixnet for a service, a developer must analyze the service extensively to identify personal information on all whitelisted pages. Using the simplistic approach that examines all responses for the data and replaces them is problematic. For example, if Mixnet replaces a users' real name in a newspaper text, one can conclude that the session-cookie of a user with the corresponding name was used, which would be a leak of personal data. Therefore, it is necessary to define precisely which parts of a response Mixnet should clean up for each service. A cleaner solution would be to find a generic way that interprets, for example, an HTML response and cleans up data based on the HTML structure. In addition to the previous problem, we need to keep track of updates on the service side. When a service adds new paths to its page, these must be propagated to the respective white-/blacklist.

Use Cases. As a field of application for Mixnet, we have limited ourselves to read-only subscription-based services with a paywall. With personalized services such as Facebook or LinkedIn, the social network and the user's profile are at the forefront of the experience, so it would not make sense to mix sessions. Nevertheless, we could extend the application area to services that require registration for full use of their service but do not have a paywall. Examples of this would be read-only proxies of Instagram or Twitter. However, to continue to achieve the primary goal of enhancing privacy, interaction with the social network and generating content with the user's profile should be avoided entirely and Mixnet should be used only to consume the content. The situation would be similar for services like Spotify or Netflix. Suppose we applied the current concept of Mixnet to a read-only version of Spotify, where interactions like creating a list or liking a song are prevented. Central functions like personalized lists and suggestions would be omitted, but the service would not learn anything about the users. If we want to keep interactions like liking songs, one approach would be to store them directly in the enclave. This information would not be forwarded to Spotify but could be attached to a response before returning it to the user. We leave the elaboration of such an approach for future work.

At this point, we would like to emphasize that MIXNET could also be used by a service provider itself, or in collaboration with it, to provide more privacy for its users. This would also ease the problem of personal data, as the design and implementation of a service would be exposed.

7.1 Future Work

We propose three possible directions for future work.

Scalability. The throughput of MIXNET implemented on a single enclave is around 500 requests per second. To scale the system, one could use the Integritee framework by running MIXNET on multiple Integritee workers. On the one hand, this would require a load balancer to distribute the HTTP requests to the different enclaves; on the other hand, the workers would have to exchange their cookie storage with each other so that an accurate session mixing can occur. In addition, future work could benefit from the upcoming Intel SGX2 version, which allows more threads per enclave than were available to us. In our setup with the current Intel SGX, the number of threads was limited to 8 threads running simultaneously, as SGX maps threads directly to logical processors.

Browser Extension. We propose implementing MIXNET as a browser extension instead of a web page. This would allow MIXNET to withstand a stronger adversary model. An implementation as a browser extension would provide access to numerous browser functionalities such as Firefox's webRequest API [33], which could be used to exceptionally route all requests originating from a page to MIXNET. MIXNET can then process, generalize, block or forward these requests. This would also include third-party requests, which may be sent for tracking purposes. The problem of dynamic URL creation that we encountered with NZZ would also become obsolete. In addition, MIXNET could do its part against fingerprinting since it has access to all requests originating from a page and therefore knows the initiator of a request. MIXNET could then block requests from certain initiators. We already mentioned that we could append custom javascript or rewrite existing javascript to solve the path rewriting problem, but MIXNET would have more capabilities as an extension. For instance, MIXNET could also get access to the browser storage, which allows accessing cookies where the `HttpOnly` flag is set. Accessing such cookies by javascript is forbidden. Having access to all cookies would make the initial transfer of the session cookie from the user to MIXNET unnecessary. Further possibilities and an evaluation of such an approach are left for future work.

Website Analysis. We have already suggested filtering out personal data based on the HTML structure. Another approach would be to integrate a tool for website analysis, such as needed for search engine optimization

(SEO), or an authenticated web crawler. The main purpose of this tool would be to analyze links and content on a website and find user-related links and content. With an automated analysis by such a tool, the manual effort for implementing services could be reduced and react to changes in the site structure. Another approach would be to analyze websites in greater depth and detect possible interactions, such as a ”like” for a song on Spotify. Mixnet could then be extended to more personalized services such as Spotify by preventing such interactions.

Chapter 8

Related Work

User privacy is becoming an increasingly important issue in our society. At the same time, there is a growing need to know whom we are trusting with our data and that it is safe to use this service. There are many approaches to improve privacy by avoiding tracking on the web, be it through privacy-enhancing tools [60] or anonymous communication through services like Tor [30] or the use of mix networks [36, 56]. TEEs are increasingly used to provide secure, trustworthy applications. Frameworks such as Integritee [16] or Enarx [5] simplify running applications on TEEs. The range of applications using TEEs includes, among others, biometric authentication [41], secure payments [52] and blockchain wallets [6], or end-to-end encryption between enclaves [23]. With Mixnet, we extend the possibilities to enhance privacy using the concept of mix networks and TEEs.

The Nym Network [46] takes a similar approach to Mixnet. It also uses the concept of mix networks and generalizes requests to a service provider, thus preventing the service provider from concluding the user who sent the request. In contrast to Mixnet, the Nym Network is based on a blockchain-based solution. In the Nym blockchain, among other things, a record is stored of which user is authorized for which service. However, the Nym Network does not forward the requests with real user identities but uses anonymous credentials [43]. This requires service providers to cooperate with the Nym Network since the service providers must accept these anonymous credentials. Mixnet, on the other hand, does not require cooperation of the service provider. More than 100 papers use the concept of mix networks. An analysis of different applications can be found in the article: "Mix Networks: Existing Scenarios and Future Directions on Security and Privacy" [36].

The two papers, DelegaTEE [54] and TEEvil [57], work with the concept of "account sharing" using TEEs. DelegaTEE deals with delegating work, such as making payments with another person's account or processing mails for them. TEEvil, on the other hand, takes the approach that one can lend his

identity to another user in exchange for money. In both approaches, a policy defines which actions are possible. The policy-based approach would be a way to extend Mixnet to personalized services by allowing or prohibiting specific actions.

Chapter 9

Conclusion

This thesis presents Mixnet, a system that enhances privacy for users of subscription-based services. Mixnet obfuscates the connection between server request and the user who made it. It does so while maintaining the service's ability to restrict access to paying customers. The main idea behind Mixnet is mixing subscription-related cookies and generalizing requests. Mixnet performs this tasks in an isolated runtime using TEEs.

Our implementation and experiments show that Mixnet can be applied to different real-world applications, such as newspapers or streaming services, and works without cooperation with the service provider. Before a request is forwarded to a service, Mixnet removes identifying headers or replaces them with generalized default headers. While doing so, Mixnet replaces the subscription-related cookie with the cookie of an arbitrary Mixnet user. To restrict access to paying customers, Mixnet keeps track of which user submitted which cookie and regularly checks the validity. Mixnet prevents displaying session-related personal data of one user to another user. It does so by blocking access to user-specific pages based on a white-/blacklist logic and detecting and replacing personal data embedded in pages. The selected approach works without cooperation with the service provider.

The throughput of Mixnet settles between 500 and 600 requests per second, and the overhead of using Mixnet remains within reasonable limits. Streaming via Mixnet adds on average only about 50 ms overhead per request. The chosen approach is thus even suitable for streaming. For newspapers the times for a full-page load are even shorter since only functionally relevant requests are forwarded. Mixnet also works with other privacy-enhancing tools, and using them side-by-side is an effective way to protect user privacy.

Appendix A

Performance Results

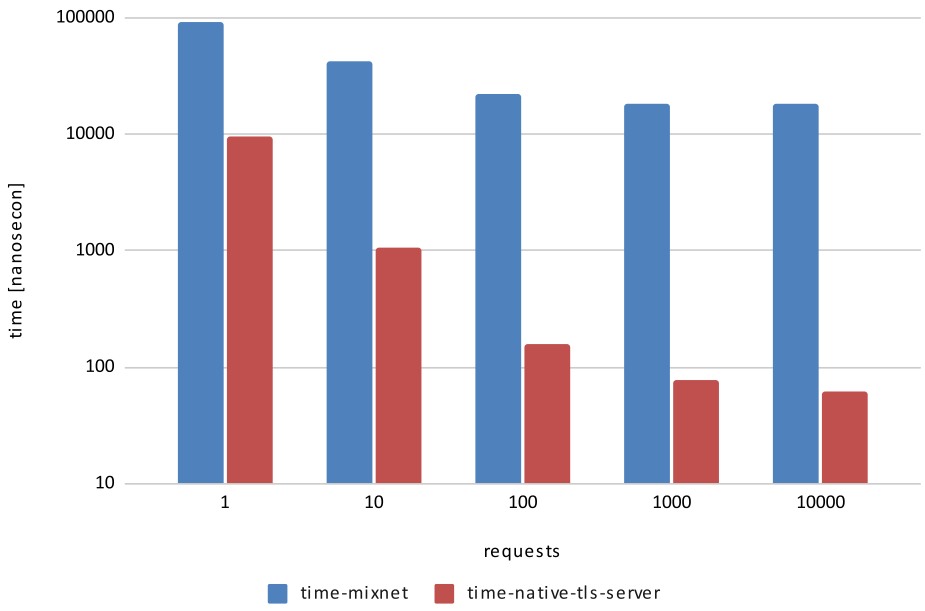

Figure A.1: Comparison of mean latencies (in nanoseconds) for requests send to MIXNET and the native TLS server by a single user

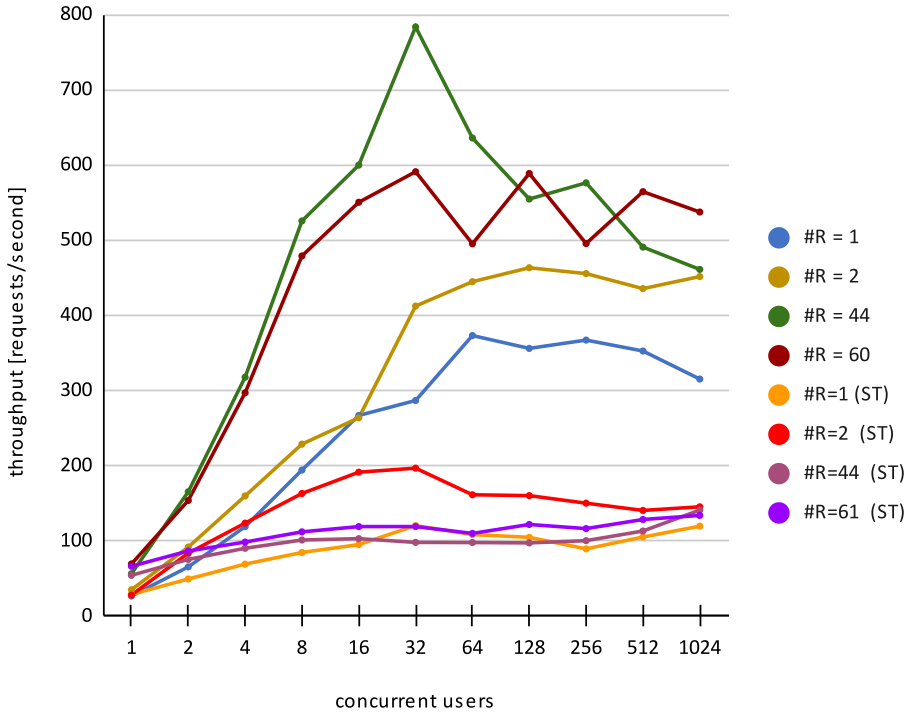

Figure A.2: Throughput comparison of multithreaded vs singlethreaded version of Mixnet

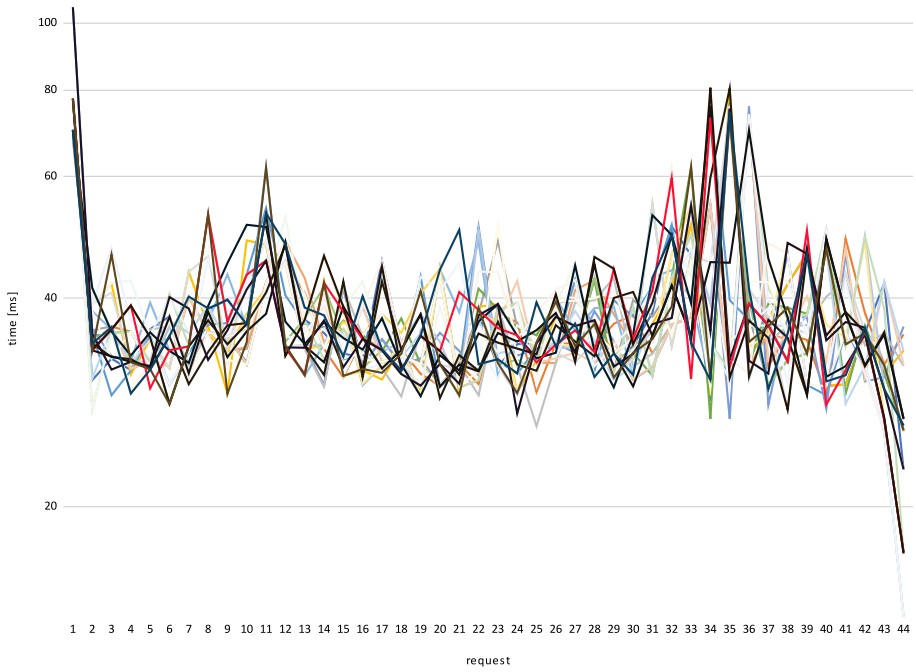

Figure A.3: Time lapse of latency per request via MIXNET for 32 concurrent users sending 44 requests each.

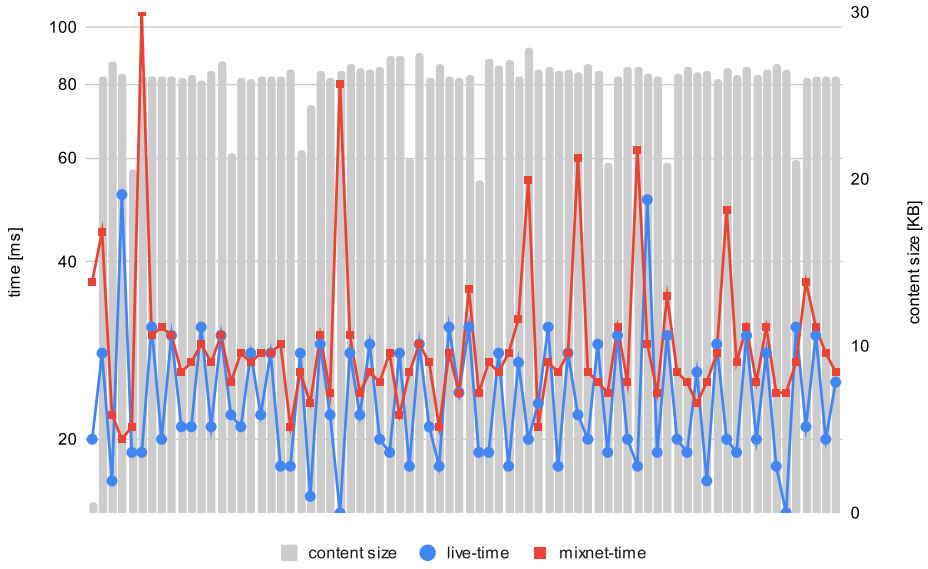

Figure A.4: Round-trip times (RTT) of audio packages while streaming Zattoo via MIXNET and directly.

Bibliography

[1] apache/incubator-teaclave-sgx-sdk: Apache Teaclave (incubating) SGX SDK helps developers to write Intel SGX applications in the Rust programming language, and also known as Rust SGX SDK. URL: https://github.com/apache/incubator-teaclave-sgx-sdk.

[2] Attestation Service for Intel® Software Guard Extensions (Intel® SGX): API Documentation. URL: https://api.trustedservices.intel.com/documents/sgx-attestation-api-spec.pdf.

[3] Attestation Services for Intel® Software Guard Extensions. URL: https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/attestation-services.html.

[4] Beaker · PyPI. URL: https://pypi.org/project/Beaker/.

[5] Enarx — Enarx. URL: https://enarx.dev/.

[6] Ethereum Wallet in a Trusted Execution Environment / Secure Enclave — by Sascha Thomsen — weeve's World — Medium. URL: https://medium.com/weeves-world/ethereum-wallet-in-a-trusted-execution-environment-secure-enclave-b200b4df9f5f.

[7] Facebook Reports Fourth Quarter and Full Year 2014 Results. URL: https://investor.fb.com/investor-news/press-release-details/2015/Facebook-Reports-Fourth-Quarter-and-Full-Year-2014-Results/default.aspx.

[8] Facebook's Top Open Data Problems - Meta Research — Meta Research. URL: https://research.facebook.com/blog/2014/10/facebook-s-top-open-data-problems/.

[9] GDPR - REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679&from=DE.

[10] Github - scs/ma-thesis-no-data-collection. URL: https://github.com/scs/ma-thesis-no-data-collection.

[11] Home — Substrate. URL: https://substrate.io/.

[12] HTTP response status codes - HTTP — MDN. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status.

[13] http-rs/route-recognizer: Recognizes URL patterns with support for dynamic and wildcard segments. URL: https://github.com/http-rs/route-recognizer.

[14] Integritee Lightpaper. URL: https://uploads-ssl.webflow.com/60c21bdfde439ba700ea5c56/612892db018a36f054100b4d_Integritee%20AG%20Lightpaper.pdf.

[15] integritee-network/worker at 5f7b47093f7aee10c0f36fc1abaf0003c4f99701. URL: https://github.com/integritee-network/worker/tree/5f7b47093f7aee10c0f36fc1abaf0003c4f99701.

[16] Integritee – Unchain the value of sensitive data. URL: https://integritee.network/.

[17] Intel® Software Guard Extensions. URL: https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/overview.html.

[18] Mitigating Cross-site Scripting With HTTP-only Cookies — Microsoft Docs. URL: https://docs.microsoft.com/en-us/previous-versions//ms533046(v=vs.85)?redirectedfrom=MSDN.

[19] Netezza and IBM Cloud Pak for Data: A knockout combo for tough data - Journey to AI Blog. URL: https://www.ibm.com/blogs/journey-to-ai/2020/06/netezza-and-ibm-cloud-pak-a-knockout-combo-for-tough-data/.

[20] NZZ – Neue Zürcher Zeitung — Aktuelle News, Hintergründe & mehr. URL: https://www.nzz.ch/.

[21] OneLog. URL: https://consent.onelog.ch/?lang=de#/.

[22] Privoxy - home page. URL: https://www.privoxy.org/.

[23] project-oak/oak: Meaningful control of data in distributed systems. URL: https://github.com/project-oak/oak.

[24] rustls/rustls: A modern TLS library in Rust. URL: https://github.com/rustls/rustls.

[25] six-ddc/plow: A high-performance HTTP benchmarking tool with real-time web UI and terminal displaying. URL: https://github.com/six-ddc/plow.

[26] Sweet32: Birthday attacks on 64-bit block ciphers in tls and openvpn. URL: https://sweet32.info/.

[27] Tages-Anzeiger — Aktuelle Nachrichten und Hintergründe. URL: https://www.tagesanzeiger.ch/.

[28] tempesta-tech/tls-perf: TLS handshakes benchnarking tool. URL: https://github.com/tempesta-tech/tls-perf.

[29] tokio-rs/mio: Metal IO library for Rust. URL: https://github.com/tokio-rs/mio.

[30] Tor Project — Anonymity Online. URL: https://www.torproject.org/.

[31] Trackmenot. URL: http://trackmenot.io/.

[32] tumpio/requestcontrol: A Firefox extension. URL: https://github.com/tumpio/requestcontrol.

[33] webrequest - mozilla — mdn. URL: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/webRequest.

[34] What is Integritee? A full technical guide. — by Vlady Limes — Polkadot Ecosystem PromoTeam — Medium. URL: https://medium.com/polkadot-ecosystem-promoteam/what-is-integritee-a-full-technical-guide-75de9c3cc7b9.

[35] Zattoo - Streaming Live TV from any device: with over 100 TV channels! URL: https://zattoo.com/ch/en.

[36] Khaleel Ahmad and Afsar Kamal. Mix networks: Existing scenarios and future directions on security and privacy. *Recent Patents on Engineering*, 14(3):310–323, 2020. URL: https://www.ingentaconnect.com/content/ben/eng/2020/00000014/00000003/art00006, doi:doi:10.2174/1872212114666191223125619.

[37] Rami Al-Rfou', William Jannen, and Nikhil Patwardhan. Trackmenot-so-good-after-all. 11 2012. URL: http://arxiv.org/abs/1211.0320.

[38] Xavier Amatriain. Big & personal: Data and models behind netflix recommendations. BigMine '13, page 1–6, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2501221.2501222.

[39] Nimrod Aviram, David Adrian, J Alex Halderman, Sebastian Schinzel, Juraj Somorovsky, Nadia Heninger, Maik Dankel, Jens Steube, Luke Valenta, Viktor Dukhovni, Emilia Käsper, Shaanan Cohney, Susanne Engels, Christof Paar, and Yuval Shavitt. Drown: Breaking tls using sslv2. 2016. URL: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/aviram.

[40] Adam Barth. HTTP State Management Mechanism. RFC 6265, April 2011. URL: https://www.rfc-editor.org/info/rfc6265, doi:10.17487/RFC6265.

[41] Abhilasha Bhargav-Spantzel. Trusted execution environment for privacy preserving biometric authentication. *Intel Technology Journal*, 18(4):162 – 177, 2014. URL: https://search.ebscohost.com/login.aspx?direct=true&db=buh&AN=97377859&site=ehost-live.

[42] Karthikeyan Bhargavan and Gaëtan Leurent. On the practical (in-)security of 64-bit block ciphers: Collision attacks on http over tls and openvpn. *Proceedings of the ACM Conference on Computer and Communications Security*, 24-28-October-2016:456–467, 10 2016. doi:10.1145/2976749.2978423.

[43] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques: Advances in Cryptology*, EUROCRYPT '01, page 93–118, Berlin, Heidelberg, 2001. Springer-Verlag.

[44] Victor Costan and Srinivas Devadas. Intel sgx explained. Cryptology ePrint Archive, Report 2016/086, 2016. https://ia.cr/2016/086.

[45] Jinhua Cui, Jason Zhijingcheng Yu, Shweta Shinde, Prateek Saxena, and Zhiping Cai. SmashEx: Smashing SGX Enclaves Using Exceptions. *Proceedings of the ACM Conference on Computer and Communications Security*, pages 779–793, 11 2021. doi:10.1145/3460120.3484821.

[46] Claudia Diaz, Harry Halpin, and Aggelos Kiayias. The Nym Network The Next Generation of Privacy Infrastructure. 2021. URL: https://nymtech.net/nym-whitepaper.pdf.

[47] Dmitry Chestnykh. *Password authentication for web and mobile apps*. 2020.

[48] Tatiana Ermakova, Benjamin Fabian, Hft Leipzig, Benedict Bender, and Kerstin Klimek. Web Tracking-A Literature Review on the State of Research. URL: http://hdl.handle.net/10125/50485.

[49] Richard Ford and Michael Howard. Man-in-the-Middle Attack to the HTTPS Protocol. Technical report, 2009. doi:10.1109/MSP.2009.12.

[50] David Greene, Eff Senior, Staf Attorney, and Katitza Rodriguez. Nsa mass surveillance programs unnecessary and disproportionate. URL: https://necessaryandproportionate.org/text.

[51] Michael Lesk. Big data, big brother, big money. *IEEE Security Privacy*, 11(4):85–89, 2013. doi:10.1109/MSP.2013.81.

[52] Joshua Lind, Ittay Eyal, Florian Kelbert, Oded Naor, Peter R. Pietzuch, and Emin Gün Sirer. Teechain: Scalable blockchain payments using trusted execution environments. *CoRR*, abs/1707.05454, 2017. URL: http://arxiv.org/abs/1707.05454, arXiv:1707.05454.

[53] Jamie R Lund. The End of Forgetting and "Administrative Rights" to Our Online Personas Personas. Technical Report 2, 2012. URL: http://kb.iu.edu/data/aorq.html.

[54] Sinisa Matetic, Moritz Schneider, Eth Zurich, Andrew Miller, Ari Juels, Cornell Tech, Srdjan Capkun, Sinisa Matetic ETH Zurich Moritz Schneider ETH Zurich Andrew Miller, and Ari Juels Cornell Tech Srdjan Capkun ETH Zurich. DelegaTEE: Brokered Delegation Using Trusted Execution Environments. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/matetic.

[55] Eli Pariser. *The filter bubble: What the Internet is hiding from you*. Penguin UK, 2011.

[56] Ania M Piotrowska, Jamie Hayes, Sebastian Meiser, George Danezis, Tariq Elahi, and K U Leuven. The Loopix Anonymity System. URL:

www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/piotrowska.

[57] Ivan Puddu, Daniele Lain, Moritz Schneider, Elizaveta Tretiakova, Sinisa Matetic, Srdjaň Capkun, and Eth Zurich. TEEvil: Identity Lease via Trusted Execution Environments. URL: https://arxiv.org/abs/1903.00449.

[58] Jaak Randmets. An Overview of Vulnerabilities and Mitigations of Intel SGX Applications. URL: https://cyber.ee/research/reports/D-2-116-An-Overview-of-Vulnerabilities-and-Mitigations-of-Intel-SGX-Applications.pdf.

[59] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018. URL: https://www.rfc-editor.org/info/rfc8446, doi:10.17487/RFC8446.

[60] A. Ruiz-Martínez. A survey on solutions and main free tools for privacy enhancing Web communications. *Journal of Network and Computer Applications*, 35(5):1473–1492, 9 2012. doi:10.1016/J.JNCA.2012.02.011.

[61] Pierangela Samarati and Latanya Sweeney. Protecting Privacy when Disclosing Information: k-Anonymity and Its Enforcement through Generalization and Suppression. URL: https://epic.org/wp-content/uploads/privacy/reidentification/Samarati_Sweeney_paper.pdf.

[62] Niklas Schmucker. Web tracking. In *SNET2 Seminar Paper-Summer Term*, volume 2011. Citeseer, 2011. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.474.8976&rep=rep1&type=pdf.

[63] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowlege-Based Systems*, 10(5):557–570, 10 2002. URL: www.worldscientific.com, doi:10.1142/S0218488502001648.

[64] Sean Turner and • Ieca. Standards Editor: Barry Leiba • barryleiba@computer.org How Did We Get Here? Transport Layer Security. Technical report, 2014. URL: www.computer.org/internet/, doi:10.1109/MIC.2014.126.

[65] Nico Weiner. Social networks evolving into service platforms-the Facebook-case from abusiness model viewpoint. URL: http://bits.blogs.nytimes.com/2008/07/03/what-is-facebook-worth-part-37/.