Information Systems 97 (2021) 101702

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is

Enterprise-grade protection against e-mail tracking

Benjamin Fabian [a,*], Benedict Bender [b], Ben Hesseldieck [c], Johannes Haupt [c], Stefan Lessmann [c]

[a] Chair of E-Government, IT-Security and IT Management, Technical University of Applied Sciences Wildau, Hochschulring 1, 15745 Wildau, Germany
[b] Chair of Business Informatics, University of Potsdam, August-Bebel-Str. 89, 14482 Potsdam, Germany
[c] School of Business and Economics, Humboldt-Universität zu Berlin, Spandauer Str. 1, 10178 Berlin, Germany

ARTICLE INFO

Article history:
Received 3 April 2019
Received in revised form 8 December 2020
Accepted 9 December 2020
Available online 17 December 2020
Recommended by Dennis Shasha

ABSTRACT

E-mail tracking provides companies with fine-grained behavioral data about e-mail recipients, which can be a threat for individual privacy and enterprise security. This problem is especially severe since e-mail tracking techniques often gather data without the informed consent of the recipients. So far e-mail recipients lack a reliable protection mechanism.

This article presents a novel protection framework against e-mail tracking that closes an important gap in the field of enterprise security and privacy-enhancing technologies. We conceptualize, implement and evaluate an anti-tracking mail server that is capable of identifying tracking images in e-mails via machine learning with very high accuracy, and can *selectively* replace them with arbitrary images containing warning messages for the recipient. Our mail protection framework implements a selective prevention strategy as enterprise-grade software using the design science research paradigm. It is flexibly extensible, highly scalable, and ready to be applied under actual production conditions. Experimental evaluations show that these goals are achieved through solid software design, adoption of recent technologies and the creation of novel flexible software components.

1. Introduction

E-mail tracking is used to collect data on e-mail recipients in a similar way that web tracking is used to collect data on website visitors. Web and e-mail tracking are popular marketing tools due to the increasing importance of accurate customer data for business success [1,2]. Analysis of customer data is used to personalize offerings and marketing layouts for an optimized market position as well as an advantage in product pricing [3]. Modern e-mail tracking methods allow the sender to determine how often an e-mail was opened, the device used to read the e-mail, which links were clicked, and the location and time when the recipient opened an e-mail [4,5]. In this context, the technology to measure user behavior is a competitive advantage in online marketing. Tracking data are so valuable that companies specialize in their gathering and use selling of aggregated-data as a business model [6] or offering e-mail tracking as a service.

Despite its use in marketing applications, e-mail tracking is a serious privacy and security threat for end-users and organizations. In contrast to cookie-based web tracking, e-mail tracking is often used by first- and third parties to collect data without permission or knowledge of the e-mail recipient, while the e-mail address of the recipient serves as a personal identifier. Beyond its legal usage, hackers and criminal spammers use e-mail tracking to determine if an e-mail account is active and whether the owner opens attachments, which paves the way for system intrusions [7]. Moreover, spam as modern crime poses several challenges in detection and prevention [8]. Despite the known risk and higher public scrutiny of the collection of user data, recent regulation initiatives such as for web tracking elements do not exist for e-mail tracking approaches [9]. In addition, the technical process of e-mail tracking, which does not require cookies, makes it difficult to block without affecting the user experience and there exist no popular applications similar to ad blockers that disable e-mail tracking on the technical level.

Prior research on e-mail tracking examined the principles [4,5] as well as the usage across regions (Bender, Fabian, Haupt, & Lessmann; [10]) and the actual employment of data gathered through e-mail tracking [11]. In Haupt et al. [12], we examined the potential of machine-learning to identify tracking elements in e-mail communication. These earlier results indicate that accurate detection is possible, which may be seen as a first step towards tracking prevention and privacy protection. Achieving the latter in an enterprise context, however, requires a more holistic approach that considers real-world requirements related

* Corresponding author.
E-mail addresses: benjamin.fabian@th-wildau.de (B. Fabian), benedict.bender@wi.uni-potsdam.de (B. Bender), b.hesseldieck@gmail.com (B. Hesseldieck), johannes.haupt@remerge.io (J. Haupt), stefan.lessmann@hu-berlin.de (S. Lessmann).

https://doi.org/10.1016/j.is.2020.101702
0306-4379/

to the efficiency and scalability of a detection engine and empirical insights how such a detection system behaves under realistic deployment conditions.

In this paper, an extension and culmination of our previous research, we close this gap in the field of privacy-enhancing technologies by conceptualizing and implementing a novel protection framework against e-mail tracking in an enterprise context. We provide an answer to the question how a reliable and efficient anti-tracking mechanism can be realized for enterprise-level environments. We realize a professional anti-tracking mail server that is capable of identifying tracking images in e-mails via machine learning and selectively replace them to mitigate tracking. Our mail protection framework is developed as enterprise-grade software, which is flexibly extensible, highly scalable, and ready to be applied in actual production conditions. Experimental evaluation shows that this is achieved through corresponding choices regarding technologies and the creation of a solid software design. To enable practical adoption and also facilitate future research, we provide our software as open source on GitHub [13].

Our study follows the paradigm of design science research [14]. The structure of this article is aligned with the major steps of this research method, starting with the problem identification and motivation in this introductory section. In the next section, the required research background is presented in order to prepare the objectives and requirements in the third section. Then, the design and development process of our software artifact is elaborated, followed by a demonstration and thorough experimental evaluation. Finally, major contributions, limitations and future work are discussed.

2. Background and related work

To provide an overview of e-mail tracking methodology and how it affects user privacy, we first discuss the e-mail tracking process and detail how and what kind of information can be captured about mail recipients. This analysis was initially published in Bender et al. [15] and provides a foundation for designing effective countermeasures in the current article.

The tracking process (Fig. 1) starts with the preparation and sending of an HTML-based e-mail, which includes a tracking-image reference. This mail passes several mail transfer agents (MTAs) until it reaches the receiver's MTA. When the recipient opens the e-mail with a tracking image, the mail client requests the image from the referenced web server, which logs this request and provides the image to the client. Log analysis at the web server allows information to be deducted on the recipient's e-mail reading behavior.

In order to assess the extent of primary (i.e., direct) information available to a tracker, we constructed a prototypical tracking environment, which includes an Apache webserver to log data relevant to e-mail tracking. The entries in the server log file provide seven major pieces of information: (1) the Internet Protocol (IP) address of the host that requests the image file, (2) the date and time of the file request, (3) the request itself, which includes the URL and GET variables, (4) the status code of the request, (5) the amount of bytes that have been sent in response, (6) the referrer URL from the client, and (7) a string characterizing the user agent. Furthermore, when a file is requested multiple times (i.e., it generates multiple entries), it allows information to be derived with respect to a user's reading behavior. In our test environment, a new log entry was created every time an e-mail was opened.

With respect to secondary (i.e., indirect) information, the first possibility is to induce the fact that the user read or at least opened the e-mail. The existence of multiple log entries indicates that the e-mail has been opened multiple times. The combination of multiple entries for one mail, as well as multiple entries from one user for different mails, provides insight into the recipient's e-mail reading behavior, also across devices.

The usage of IP geolocation gives indications on possible user locations also allows the detection of forwarded mails. Special log entries allow one to determine whether an e-mail has been printed [16]. It is also possible to gather information about the user environment by analyzing the user agent string, which is part of a log entry [17]. Based on a reverse lookup of an IP address, a log entry may also help determine a user's affiliation to a company or institution.

We now discuss related scientific work. Publications on the general issue of web tracking are numerous, ranging from its usage [2,18–22] to its detection [23–26] and prevention [24,27–29]. Some publications on web tracking hint at the opportunity of applying tracking mechanics also to HTML-based e-mail [6,25,27,30,31]. However, none of these studies explore this possibility in depth.

E-mail tracking itself is mainly investigated and utilized in the context of at least four research fields: marketing, malicious e-mails, spam, and privacy. The first research stream explores the effects of *e-mail marketing* on the recipients and its optimization [32–35], while utilizing data gathered through e-mail tracking. Publications from Bonfrer and Drèze [36] as well as from Hasouneh and Alqeed [37] investigate the tracking technology and process from a marketing viewpoint and emphasize the importance of tracking newsletters and any other e-mail marketing communication. The Direct Marketing Association (DMA) annually releases research reports about e-mail tracking and its impact on online marketing [3].

The second research avenue is the field of *malicious e-mail* content or attachments, where e-mail tracking is used to analyze the mechanics and velocity of spreading viruses or malicious programs [38,39]. Third, the issue of *spam* is necessarily an important issue in e-mail research, but also e-mail tracking has connections to it. Mechanics of e-mail tracking are used to identify the origin of spam mail in order to add the sender to blacklists [40–42,42]. The fourth field this article draws upon is *privacy*. Protecting privacy against tracking that aims to expose the end-user's personal information to marketers is the main focus of this research direction. Various environments are investigated, such as the company level, the browser, and e-commerce [4,5,43–45]. While prior studies considered only the technical feasibility of e-mail tracking, Bender et al. [11] also showed the actual use of customer-behavior data derived from such tracking.

Although the issue of e-mail tracking is analyzed in literature, no effort has been devoted to the creation of a software solution for end-users, nor are prevention methods explored in depth, with the exception of Bonfrer and Drèze [36] and Englehardt et al. [4]. The former focused on investigating the current status of e-mail tracking and the design space for effective countermeasures and also provided initial drafts and evaluation of a machine-learning based detection model. Englehardt et al. [4] empirically surveyed the current landscape of mail tracking with a particular emphasis on third-party trackers. Furthermore, they assessed the incompleteness of existing defense solutions, and briefly outlined a novel anti-tracking strategy based on filter lists. Additionally, Haupt et al. [12] benchmarked various machine-learning approaches and thereby guided the design of effective and reliable detection mechanisms.

These results provide the motivation for the countermeasures and detection model utilized by this study's software artifact. Sharing the goal of preserving e-mail end-users privacy, this article aims to develop a comprehensive and enterprise-grade countermeasure solution.

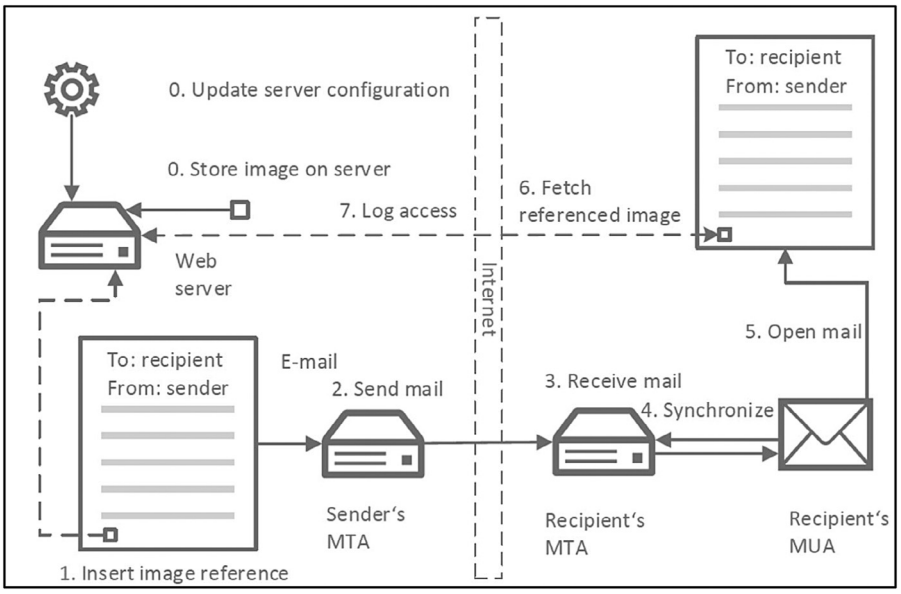


Fig. 1. E-Mail Tracking Principle.

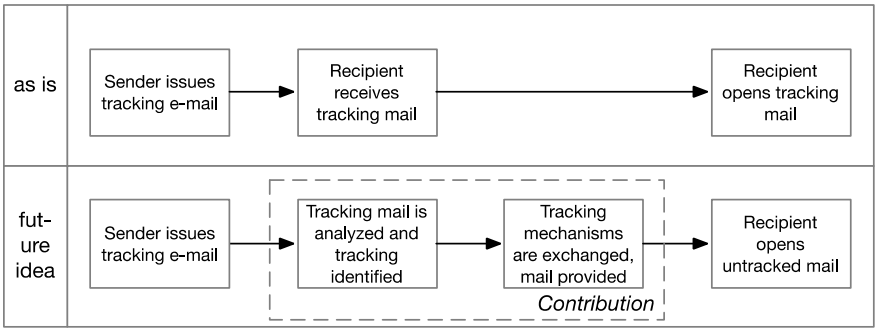


Fig. 2. Comparison of status quo and the desired tracking prevention.

3. Solution objectives

The objective of the study is to integrate prior research on the characteristics of e-mail tracking elements and their automated detection and develop a reliable and efficient enterprise-level environment to block e-mail tracking with minimal impact on user experience. Visualizing the functionality of the envisioned system, Fig. 2 shows the (unprotected) *as-is* and future process scenarios for tracking prevention, including the major functional components (detection and replacement). The corresponding system requirements can be found in Table 1 and are discussed in the following.

Our previous work identified different anti-tracking strategies. These can be categorized in preventive and deceptive approaches. Preventive approaches aim strive to hide or modify the information, so that obtained information is selected, modified, or deliberately corrupted information. In contrast, selective countermeasures are based on identifying and blocking tracking elements within an e-mail (Fig. 3). The idea is to categorize the referenced images into content-providing and tracking images. Assuming that images of the former category do not provide tracking functionality, they remain untouched, whereas tracking images are removed from the e-mail. The strength of the selective concept is the combination of systematic prevention of tracking images while preserving the full user-experience. The possible pitfall is the risk of misidentification. The solution will work only well with accurate algorithms for tracking-image identification.

Fig. 3. Selective Blocking Approach.

Based on the discussion in [15], we concentrate on a selective identify-and-block strategy since these are assumed to provide the best balance between preventing user tracking and sustaining user experience. Accordingly, we concentrate on this approach for the proposed solution. Therefore, the first requirement for the solution is to realize the concept of selective blocking (requirement 1).

Following the selective blocking concept, a detection engine for tracking elements is required to achieve *accurate detection* that identifies tracking images in HTML-based mails which are then exchanged an arbitrary non-tracking image [15]. Previous studies identified relevant features and compared classification methods for tracking image detection [12]. In short, the solution is required to feature an accurate detection engine (requirement 2) as this determines the usefulness as well as the effectiveness of the solution.

Moreover, the universality (requirement 3) over different senders and time are important to provide reliable protection also even against evolving tracking mechanisms [12,15]. This involves the implementation of a detection engine, which after a training period can automatically classify previously unknown images into tracking and non-tracking categories.

Concerning modularity (requirement 4), all system components should be flexible exchangeable, including the detection engine so that the system is able to use different or additional detection algorithms or frameworks in the future. System modularity allows easy maintainability and extensibility. For example, enabling developers to plug-in new machine learning models and algorithms into the framework.

Moreover, the system needs to provide scalability (requirement 5). It is only suitable for enterprise use if appropriate performance is demonstrated and the solution is easily scalable towards high traffic. Large companies or e-mail providers deal with thousands of mails in a short timeframe. This demands an elastic solution, supporting small-scale use cases while being able to scale up to an enterprise level.

Finally, the solution should provide heterogeneous endpoint coverage. Given that user receive e-mail via different devices clients, the solution needs to provide flexible coverage (requirement 6). Previous client-side approaches showed a poor performance on this aspect [15]. In this regard, the solution should be deployable at single endpoints or at more central stages such as a gateway, a mail server, or even in the cloud.

For flexible deployment options, we decided that the implementation of the framework should not require more than 500 megabyte (MB) storage space and less than 250 megabyte of memory (both without the detection engine) since these resources can be assumed to be available in most enterprise infrastructures or cloud services, where such small configurations are usually offered for free. Further, a reasonable buffer should be planned for both metrics, in order to be prepared for extensions and peak traffic situations.

E-mail services usually do not work in real-time, therefore high performance is not extremely crucial in our application compared to other services. On the other hand, long delays could lead to limited scalability. The additional processing of e-mails by the tracking prevention system will inevitably cause some delay in e-mail delivery. We suggest a maximum threshold of five minutes for practicability reasons. The system should keep the average delay in the range of seconds, but under no circumstances exceed the five-minute threshold. Regarding scalability, the framework should be able to cope with at least 20 concurrent connections. Achieving such a degree of scalability qualifies the framework for enterprise applications, while the explicitly low resource requirements allow the system to be run and scaled up on free popular Platform-as-a-Service providers.

4. Design & development

Designing a protection solution as shown in the lower panel of Fig. 2 requires several steps. First, the tracking prevention approach needs to be defined. Second, high-level realization decisions need to be taken. Third, necessary decisions regarding architecture and implementation need to be taken.

4.1. Prevention concept

We base on Bender et al. [15] and consider the concept of selective blocking to be most suitable for combining systematic tracking prevention with good user experience (see requirement 1). The selective blocking approach follows the identify and block idea. All referenced external images in a mail need to be classified as either being tracking or non-tracking images. While non-tracking images remain, tracking images are either deleted or replaced to preserve the e-mail format.

4.2. High-level design

To provide the functionality of the selective blocking approach, the combination of tracking detection and e-mail modification is required. Providing this functionality can be realized on the client-side as well as on a server. While server- and client-based approaches share characteristics, many differences exist that are to be considered when designing a tracking prevention software solution. Table 2 gives an overview of criteria that distinguish the two paradigms to decide for the more suitable one with regard to our solution objectives.

The server-based solution is installed on the mail server and therefore tightly integrated with the central point of mail reception. Concerning the first requirement, neither client-side solutions nor server-based approaches provide sufficient protection. While server-based solutions do not yet exist, different studies revealed client-side solution to not provide sufficient protection [4, 15]. Concerning the detection engine (requirement 2,3,4) related requirements can be realized in both approaches. Considering the quality of protection, the server-based approach can profit from receiving similar mails. Tracking mechanisms can be identified by analyzing differences between similar mails as tracking approaches require uniqueness identification of e-mail recipients [5]. Therefore, slight advantages exist for the server-based solution. Concerning the necessary performance and scalability (requirement 5), central server resources are assumed to be more powerful, than local client-side devices. However, modern devices are assumed to provide necessary resources. Since the relevant functionality in bundled on the server-side, the solution is client independent. Modifications and specialized functionality are not required on the front-end (client) side. On the contrary, the client-side solution requires each device and mail client to realize the detection and modification procedures (requirement 6). If at least one mail client does not realize this, protection is insufficient. Considering the many different mail clients, platforms, and devices, the client-based approach requires extensive development and setup efforts compared to the server-based solution.

Summing up, the server-based approach fulfills the requirements and demands to a greater extent, especially in the professional context, as enterprise-infrastructures involve centrally managed mail infrastructures. We therefore focus on a server-based solution in the realization.

4.3. Software specification

For ensuring the high-quality demands of such complex software, a well-founded engineering process was applied to operationalize the design and development phase of the design science research process. A model that matched the nature of the task is the *Rapid Application Development* Model by Martin [46]. Its development lifecycle is designed to enable faster development and higher quality results than traditional process models [47].

General architecture

In correspondence with recent trends and the requirement for scalability in enterprise software development, we design our framework as a flexible *micro-service* architecture [48]. The process and data flow are shown in Fig. 4. A detailed description is provided in the subsequent sections.

A system consisting of separate services brings about additional challenges for communication and security. Use of the Docker platform [49] and its container-based system support the objectives of universality, user-friendliness, flexibility, and plugin architecture on a system level. Docker also provides a local network that can be used for communication between running containers (Fig. 5).

Table 1
System requirements.

ID	Requirement	Rationale	Category
1	Selective blocking	Selective blocking of tracking images is identified as the most suitable approach that also preserves usability [15]	Concept
2	Accuracy	The accuracy of the detection engine determines the effectiveness of the entire system [15]. Depending on misclassification, different effects may occur (false positives/false negatives) [12].	Algorithm
3	Universality	The detection performance needs to be stable over different mails senders and also over time; the system needs to recognize new forms of tracking [12].	Algorithm
4	Modularity	System components should modular, in particular easily adaptable to newer versions [12,15] or alternative algorithms.	Architecture
5	Scalability	Scalable performance of the entire system is required in order to be employed practically [15]	Architecture
6	Coverage of heterogeneous endpoints	The system needs to cover heterogeneous endpoints and e-mail clients in use [4].	Architecture

Table 2
Comparison of client- and server-side approaches.

Evaluation criteria	Requirement	Server-side solution	Client-side solutions
Sufficient solution available?	1	No (not available at all)	Not sufficient [4,15]
Detection engine	2,3,4	Similar quality in both approaches possible, modularization in both approaches possible; central solution allows for improved detection through similar mails in multiple inboxes.	
Performance	5	Central, powerful server resources available; however, due to heavy traffic scalability is important.	Local, sometimes mobile use-cases require energy-efficient solutions; modern technology platform should provide necessary resources.
Completeness of protection	6	Yes, e-mail server is central incoming point for messages.	Only if supported and active on all clients prior to receiving and opening mails.
Necessary setup		Once, no user-configuration effort	For each single endpoint, i.e., mail client.
Multi-platform support		Not required	Required

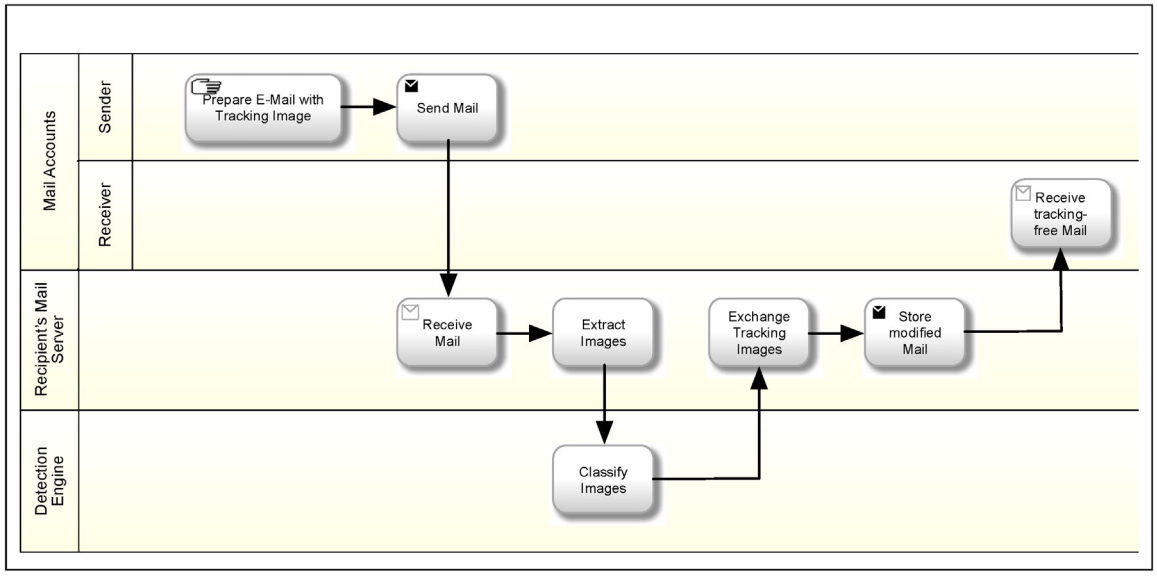


Fig. 4. Process and Data Flow.

An advantage of this software design is that the services are only loosely coupled through a communication protocol and can be easily replaced. Further advantages of the decomposition of an application into services are high system modularity, treatment of services as black-boxes, which makes the application easier to understand, and allows parallelized development, which is of special importance for large applications with multiple teams.

4.4. Technological building blocks

Node.js is a cross-platform JavaScript (JS) runtime-environment that can execute JavaScript code on the server. It was chosen for this project due to multiple reasons. First, it comes along with an active and big ecosystem that provides a lot of libraries and frameworks for uncommon issues. Second, it prevents a possible bottleneck (connecting to the detection engine) through its asynchronous I/O ability. Third, it is lightweight

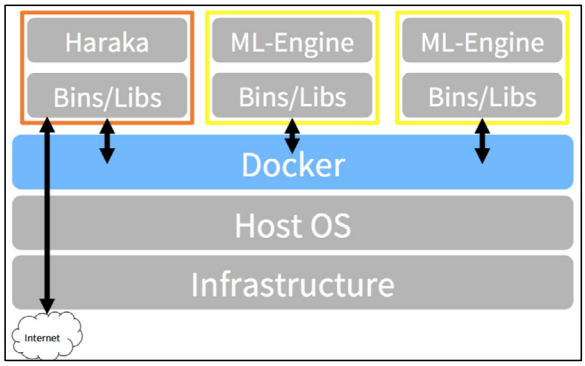


Fig. 5. Detailed Architecture, with Load-Balancing of the Detection Engine.

and runs on low resources, but is still very performant and scalable at the same time. Fourth, it considers possible open source contributions, since JavaScript and Node.js are widely known technologies with a large developer pool. Based on the project's objectives and requirements, one can see that especially the technical requirements like scalability, performance, and low resource usage could be met by utilizing Node.js. Version 8.6.0 was used for development, but compatibility with updates for the long-term supported version 8 is ensured.

Haraka is an open source Simple Mail Transfer Protocol (SMTP) Server written in JavaScript using the Node.js platform. It is built as plugin architecture around a very lightweight SMTP core, which provides software engineers to hook into the mail processing. This flexible architecture paired with the event-driven access on the SMTP processing, grants developers all options to shape the server's behavior for their needs. Haraka is selected for this software framework due to ideally matching the requirements. It has high performance, allows customized behavior, has a plugin architecture from the base on, supports hook-ins into the SMTP processing, developer friendly, comes along with security features, and foremost is it not bound to any particular e-mail service provider.

R is a widely established open source framework for statistical calculations and machine learning. Together with Python, R can be seen as a standard platform for contemporary data science. While some studies criticize R for bad memory management [50], a completely rewritten version of the R core was recently provided by Microsoft (Microsoft R Open) aiming to address such concerns. Therefore, we acknowledge that several interesting alternatives for implementing the machine-learning-based detection engine exist and continue to emerge in the form of new programming languages for data science applications or user-friendly graphical machine-learning platforms which automatically generate deployment code. With these future considerations in mind, we realize the detection engine as a plugin, which communicates with the e-mail server via HTTP using the package jug. The plug-in design makes it easy to replace the detection engine, which we currently implement in R, with some alternative technology upon need.

Docker is open source software that virtualizes an operating system (OS) for cloud applications, which are running in containers and are therefore isolated from the actual OS and from each other. It provides a lightweight layer of abstraction between the host OS and the containers, which enables certain functionalities such as setup automation and a separate local network for the containers. Docker utilizes resource isolation features of the Linux kernel to allow independent containers to be executed in a single Linux instance. This avoids the overhead of starting and maintaining a virtual machine [51]. Docker-compose is a feature of the Docker platform that allows defining and running multi-container applications. It performs the configuration, creation, and start-up process for all of the application's containers with a single command. The convenience of Docker leads to a high adoption in the software industry [51,52].

Docker was chosen for the framework due to the alignment of its features with the objectives and requirements. Especially its plugin-oriented architecture, flexibility, universality, user-friendliness, maintainability, extensibility, scalability, and security demands made Docker the platform of choice. The desired plugin architecture is supported through the container system as well as extensibility and scalability, flexibility and universality are given because Docker runs on every Linux-based OS and Windows. Security is enhanced since Docker-compose provides an internal network for containers to communicate. Only the container running the e-mail server is exposed to the Internet, while the containers with the detection engines are not. Therefore, researchers and developers not have to consider network security when creating new detection engines. With a one command setup and start the whole application ecosystem and providing scaling out of the box, boosts user experience and adoption potential of the framework.

4.5. Data flow

Fig. 1 shows the top-level data flow in the framework. It starts with the sender sending an e-mail, which is routed via an e-mail server that implements our software. The Haraka SMTP server registers the incoming mail and executes authentication checks; if they pass, it starts receiving the e-mail. After all data is received, an event is emitted and the customized tracking prevention plugin "hooks" into the processing of the e-mail. As a first step, the body and the headers of the e-mail are parsed and handed over to the e-mail extractor function. This obtains the images with all their attributes from the large HTML-string, and prepares the headers to be passed along with the images to the detection engine.

Then, the mail extractor derives additional data and returns an array consisting of all images of the e-mail. The returned array is handed over to the communication module that sends the image objects to the detection engine. The mail server is not blocked while the detection engine processes the images. Consequently, the plugin can process the next e-mail until the response from the detection engine arrives.

When the answer containing all tracking images arrives, their source needs to be replaced in the e-mail body. Since the body of the e-mail is a large HTML-string, regular expressions (regex) can be applied. If a matching string is found, it is replaced by a new image source. Finally, the e-mail body is replaced with the tracking-free version and the e-mail is forwarded to its recipient.

Executing all of these steps without intermediary checks could be wasting computing power and makes the system more error prone. For example, if an e-mail does not contain any images, it should not run through the whole process, but rather be directly passed to the mail forwarding function. The same holds if the detection engine did not find any tracking images. Fig. 6 displays the corresponding control flow of our software as UML activity diagram.

Analyzing a software system based on time, communication and execution provides an important perspective for a deeper understanding. In Fig. 7, the sequential execution and communication of the framework is shown using UML. Stick arrowheads represent asynchronous messages, and the triangle head stand for synchronous messages.

In the diagram, the process of the framework is started when an e-mail arrives. As a first step, the mail extractor module is

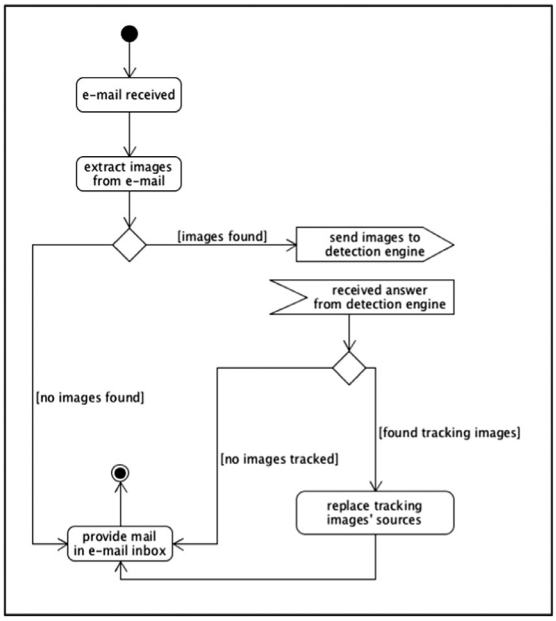


Fig. 6. UML Activity Diagram of the Software Framework.

called with the e-mail body and headers as arguments. After the function returns, it is checked whether the e-mail contained images, which are then passed via the communication module to the detection engine. When the response from the detection engine arrives, it is checked if tracking images were detected. If so, the tracking images' sources are replaced with the link to a warning image. After completing the replacement, the e-mail is forwarded to the actual recipient's inbox.

4.6. Detection engine

We build the detection engine on a machine learning classifier that identifies individual tracking images based on input features extracted from the email code [12]. The features are calculated from the HTML code, since loading the images is prohibited by the application, and fall into two groups. First, we capture the formatting of the image reference URL, since tracking image references typically show patterns of being generated and managed automatically. Indicators of these patterns are, for example, the file format or calculating statistics on the length, number of folders and changes between letters and numbers. Second, we relate each image to the other images within the same email, since separation of tracking and content management infrastructure manifests itself in distinct patterns within the URL. Tracking image links are accordingly served from different domains or have different folder structures or file formats than, for example, advertising content within the same email. All features are selected to be resilient against active manipulation by the tracker, which disqualifies absolute image size and keyword matching used in previous studies.

The detection engine is set up as a combination of the feature extraction module and any state-of-the-art classifier module to allow convenient updating of features and the classifier over time. For this study, we evaluated a logistic regression and a random forest model [53] on classification accuracy and execution time to identify the binary classifier that is best suited to classify images as "tracking" or "non-tracking" based on the proposed features. The available data consists of newsletter emails collected from 300 companies in a 20-month period from 2015 to 2017 in a controlled experiment and contains 794,519 external images within 23,602 unique emails. We tune the detection models using five-fold cross validation on a training set of emails from a 5-month period in 2015. To ensure robust performance of the classifiers out of sample, we evaluate the classifiers on 30 randomly selected companies, whose emails were excluded from the training data. To further ensure robust performance over time, we include only emails received after the training data in a 15-month period from Nov 2015 to Jan 2017. We repeat the sampling and model training process ten times and report the average results over all company samples.

The proposed selective prevention system is intended to block only specific tracking images rather than all images in an email in order to inhibit the user experience as little as possible, while ensuring a maximum level of user privacy. However, the implicit

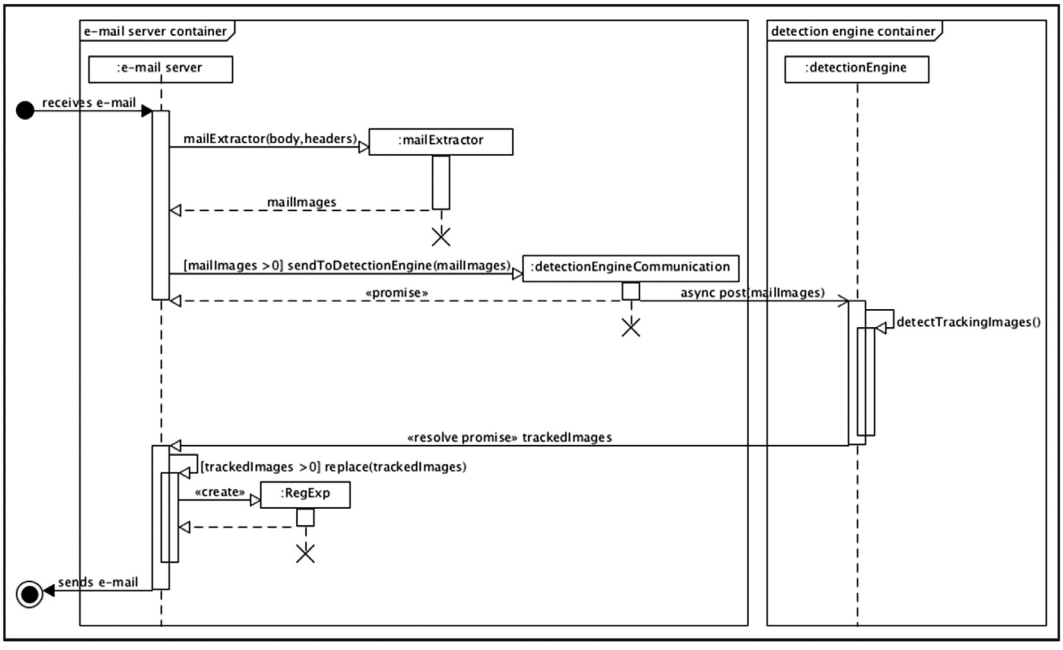


Fig. 7. UML Sequence Diagram of the Software Framework.

Table 3
Accuracy measures of classification models for tracking image detection.

	Sensitivity	Specificity	AUC
Blacklist	0.29	0.98	0.637
Logistic regression	0.98	0.72	0.972
Random forest	0.92	0.98	0.994

cost of failing to block a tracking image and allowing a breach in user privacy is higher than falsely blocking content images and impeding readability. We tune the probability threshold (for each classifier individually) on the training data set to account for the cost imbalance and assess the performance of the classifier in terms of the sensitivity and specificity, i.e., the percentage of tracking and non-tracking images that are correctly classified, respectively. For the empirical evaluation, we identify the probability threshold to be the value that maximizes the specificity of a classifier at a fixed sensitivity of at least 99.99% on the training data, based on the believe that many users have a strong preference for privacy. Future implementations could consider the sensitivity/specificity tradeoff as a user choice and set the threshold accordingly. Having fixed the sensitivity of each classifier on the training data, we compare detection engines w.r.t the mean sensitivity and specificity over the ten test sets described above and report the area-under-the-ROC-curve (AUC) for completeness (Table 3).

We observe that both statistical learning models outperform a blacklist approach based on known tracking providers and conclude that logistic regression and random forest are effective at detecting tracking images in a real-world setting. Regarding user experience, the logistic regression correctly classifies 76% of non-tracking images, while the random forest classifier correctly identifies over 99% of non-tracking images and thus leaving the email content completely intact for most emails. We adopt the random forest in our current implementation of the detection engine and stress its comparability to other state-of-the-art machine learning classifiers in terms of model complexity and prediction delay.

4.7. Scalability

Using Node.js as the underlying technology of the e-mail server provides good scalability. However, Node.js is a single threaded technology, which imposes certain restrictions. The Node.js community found a remedy in building a native cluster solution, where an orchestrating master process is spawned with the potential of starting as many workers as the system has central processing units (CPU). Haraka inherits this cluster technology.

The detection engine is a service that should be scaled when experiencing heavy load. To ensure proper usage of resources, a load balancer is placed before the detection engine containers. The load balancer distributes incoming messages from the e-mail server according to the free capacity of the detection engines. This mechanism keeps response rates of the complete system to a minimum and allows it to even handle huge amounts of traffic. If required, also the e-mail server can be multiplied. Recommended is to also place a load balancer in front of the e-mail server containers when scaling. Another feature of the system is that any scaling can be executed on startup, or even when the system is running, with one simple command (docker-compose up –scale detectionengine=2). This command starts the system with two detection engine containers as shown in Fig. 2.

In summary, the system provides all required functionalities to serve on a large scale. Due to this fact, enterprise-grade usage and ease of use are supported.

5. Demonstration and evaluation

5.1. End-user perspective

For demonstration, an excerpt of a real-world e-mail is shown in Fig. 8, an HTML newsletter from an online education platform. Its content contained tracking images that were filtered by our application.

One larger tracking image (a logo) as well as a tiny tracking pixel (at the lower left end of the e-mail) are detected. The red warning sign in Fig. 8 demonstrates how a tracking image is replaced from an end user's perspective. It also illustrates that styling of mail content remains intact. Using this technique retains user experience in contrast to the intrusive approach of preventing all image downloads.

5.2. Software complexity

An analysis of the software's complexity clarifies how efficiently an algorithm or piece of code operates, which is important in practice when system operations are time or resource critical. Accordingly, the algorithm's CPU (time) usage, memory usage, disk usage, and network usage should be taken into account. Disk and network complexities are negligible for the framework because network traffic is limited to sending from the mail server to the detection engine and back and no disk usage is implemented.

The e-mail server executes the image extraction, sending to the detection engine, and replacing of tracking images' sources. Sending the images is constant in its time complexity and linear in space complexity because the memory required depends on the amount of images in the mail body, but it has no influence on the amount of messages to be send. The mail extractor is a more complex piece of software. It needs to iterate over the HTML string to extract the images and iterate over the image objects multiple times to destruct styling, calculate similarities etc. Clearly, a decision had to be made to either keep space complexity low at the expense of time complexity or vice versa. Due to e-mail not being a time-critical service, the first option was selected so quadratic time complexity was introduced in order to achieve a logarithmic space complexity. An advantage is that the objective of running on low resources can be met even in high load situations. Delay would increase, but the memory used will stay more or less the same.

In pre-studies, we established that the detection engine is able to classify images with high speed, though there is room for performance improvements, as the experimental section will show. More time is needed for the training of the machine learning models. However, this can be conducted offline and new models can be rolled out to the detection engine in a production environment at regular intervals.

The last operation is the replacement of tracked links in the images' sources. Through utilizing a regular expression, a linear time complexity with a constant space complexity is achieved. Linear time complexity results from iterating over the HTML-string just once and constant space because the pattern as well as the replacement is stored as string. Other options would have introduced a lot of complexity to the code and probably achieved a worse result.

Overall, the complexity assessment also shows that the application is more sensitive to a large number of images in the e-mail body than to heavy traffic. But if the system is under heavy load with mails containing loads of images, the system will just slow down due to time complexity issues but will not break because of exceeding memory space.

Fig. 8. Tracking Image Replacement.

5.3. Performance experiments

Conducting performance experiments is an essential measure of the framework's suitability for production. For application in an enterprise environment, the responsible engineer needs to know how many instances and resources are needed to handle the traffic of the company.

The tests are operated using the Apache JMeter tool on an Apple MacBook Pro Mid 2017 with a 3.3 GHz Intel Core i5 (i5-7287U) and 16 GB 2133 MHz LPDDR3 RAM. Setup and running the software framework was conducted using Docker. Testing on a laptop and not a dedicated production machine serves as performance baseline. Focus is put on the analysis of multiple test settings to simulate real traffic and to inspect performance behavior for different variables. Scaling scenarios are realized through putting heavy load with a real traffic simulation on the system.

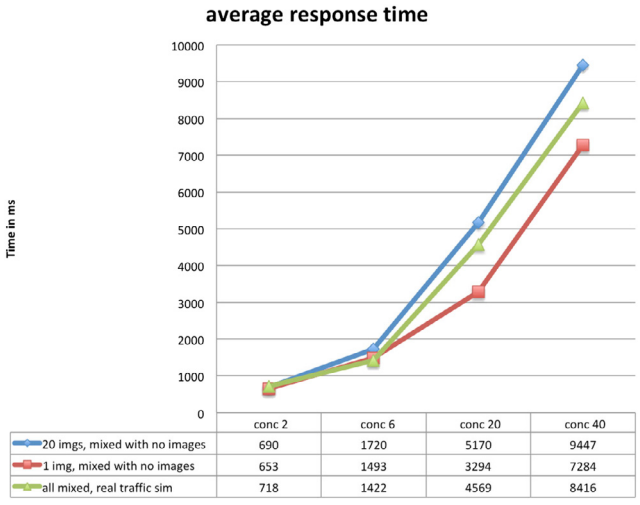
First, the response times of the system are analyzed for the combination of feature extraction plus a random forest classifier. Different e-mail contents are mixed to monitor behavior towards the number of images since the complexity analysis in the previous section indicated sensitivity towards this variable. Three tests were developed, real traffic simulation, meaning a mixture of one, twenty, and no images in the mail content. Secondly, one and no image mails mixed and finally, twenty and no images mixed. All tests were executed with a different amount of concurrent connections ranging from 2 to 40. Thereby, the test tool opens a new connection whenever another one has closed in order to keep a constant amount of connections open. For the first test plan, the framework was running on one container per service: one mail server and one detection engine.

The results show a nearly linear relationship between concurrent connections and average response time in all tests (Fig. 9). Comparing numbers, the average response time approximately doubles if the amount of concurrent connections doubles.

This result is very promising and indicates that the system has not reached its limits yet, since no exponential growth of response time was observable. Being able to handle 40 concurrent SMTP-connections with ten seconds delay on average already proves the framework to be suitable for midsize companies with this single instance setup.

Furthermore, it has to be mentioned that during all tests the system had an 1.5 to 2.0 e-mail throughput rate per second. Speed alone is not the only important indicator for the framework; also the error rate has to be considered. As previously noted, Node.js and Haraka become slower, but are not generating errors. This is supported through the tests by achieving a 0.00% error rate.

In the previous section, it was estimated that the number of images in the e-mail body impacts delivery delay due to a quadratic time complexity, which is confirmed by Fig. 9. It shows that the test with just one image has the lowest average response



	conc 2	conc 6	conc 20	conc 40
20 imgs, mixed with no images	690	1720	5170	9447
1 img, mixed with no images	653	1493	3294	7284
all mixed, real traffic sim	718	1422	4569	8416

Fig. 9. Average Response Times on a Single Instance with Mixed Traffic (in Milliseconds).

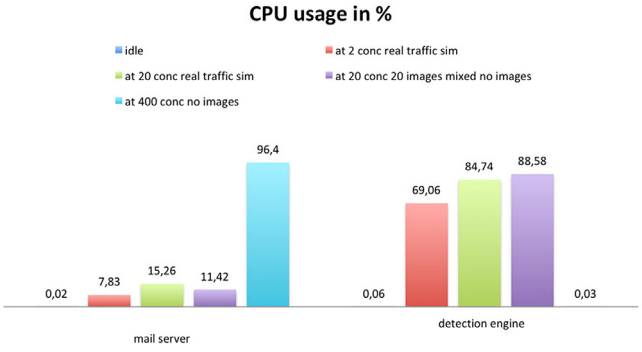


Fig. 10. CPU Usage During Experiments (in Percent).

time, while the test with twenty images resulted in the highest. Real traffic simulation is situated in between, which meets the expectation since the test sent twenty, one, and no image-containing e-mails. Ultimately, it is clear that the framework's performance is strongly influenced by the number of images in the e-mails it is processing.

Monitoring system resources is important to check if the low resource objective is fulfilled by the system. During test execution the system metrics were measured and resulted in the data shown in Figs. 10 and 11. Fig. 11 clearly reveals the CPU distribution between server and detection engine. Depending on the task, the usage share is flexibly distributed; when more cycles

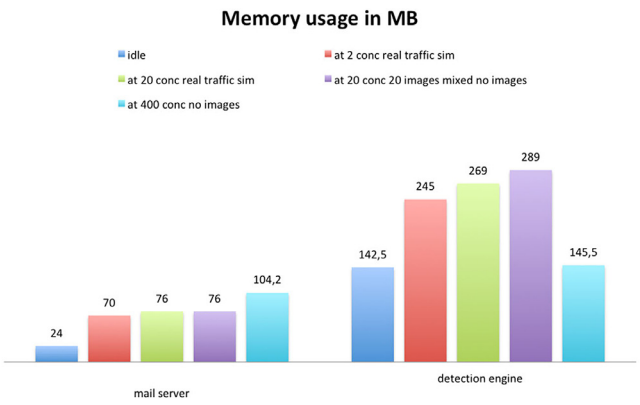
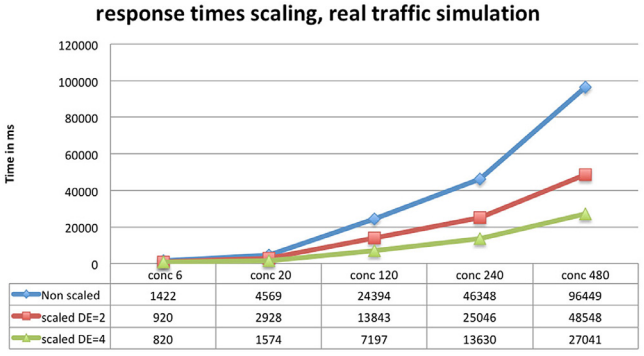
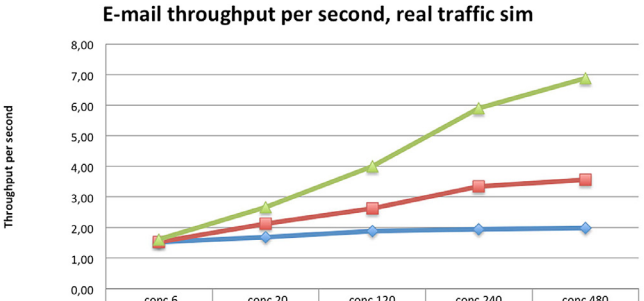


Fig. 11. Memory Usage During Experiments (in Megabytes).



	conc 6	conc 20	conc 120	conc 240	conc 480
Non scaled	1422	4569	24394	46348	96449
scaled DE=2	920	2928	13843	25046	48548
scaled DE=4	820	1574	7197	13630	27041

Fig. 12. Average Response Times on Scaling, Simulating Real Traffic (in Milliseconds).



	conc 6	conc 20	conc 120	conc 240	conc 480
Non scaled	1,53	1,68	1,88	1,95	1,99
scaled DE=2	1,53	2,13	2,62	3,35	3,57
scaled DE=4	1,62	2,67	4,01	5,90	6,88

Fig. 13. E-Mail Throughput per Second, Simulating Real Traffic.

are needed for the detection engine the mail server lowers its share. It is observable that the CPU is not fully claimed on three concurrent connections. An interesting difference exists between 20 concurrent connections with real traffic and 20 concurrent connections with a higher image load. Recalling the faster response times of the real traffic simulation from Fig. 9, this can be explained through allocation of more CPU resources to the mail server, whereas the detection engine required more CPU power on higher image load, leading to slower response times.

Fig. 11 supports this argument; the detection engine required more memory during high image loads and keeping the mail server's memory stable at 76 MB. Accordingly, the image sensitivity issue is backed through the system monitoring metrics.

In addition to the image traffic tests, a standard load test was performed to measure the general behavior of the framework when handling e-mails without images. The measured system metrics support the complexity analysis. When exposing the framework to the enormous amount of 400 concurrent connections, it reacts by using a lot of CPU cycles, but stays low on memory resources. So, the framework behaves exactly how the complexity analysis predicted. In this heavy load test the software artifact achieved a 7.6 s average response time, 0.00% errors, and an impressive throughput of 45.24 e-mails per second. These numbers show the performance of the framework when not having to handle image detection. One has to remember that these numbers were achieved running just one e-mail server- and one detection engine container.

When switching from one instance per service to multiple service containers, at first the bottleneck service needs to be identified. Looking at the previous results, it is straightforward to conclude that the framework can be just as fast as the detection engine. So, there is one issue in service-oriented systems: the whole is just as fast as its slowest part. Having loosely coupled services allows simply multiplying service instances to achieve horizontal scaling. This mechanic mitigates problems of bottleneck services. As a consequence, the detection engine was scaled up to two and four instances, and the test load was further increased through more concurrent connections.

Analysis of Fig. 12 reveals that the framework's response times are growing as a polynomial, but the effect is moderated for the scaled instances. Linear behavior as in Fig. 9 can be also observed here. Response times are linearly correlated with the number of detection engine instances and therefore increase with a smaller ratio. This behavior was expected.

What was unexpected, however, is the fact that the single detection engine system still manages to handle 480 concurrent connections with growing just to an average response time of one minute and forty seconds, which is yet an acceptable speed for asynchronous e-mail. Again, there were no errors returned for any request in the tests and even the result of the single instance framework was far away from typical sender SMTP timeouts that range between two to ten minutes (Braden [54], Section 5.3.2). Multiple detection engine instances provide no advantage for few concurrent connections, especially when put into context with the increased resource usage.

On the other hand, the horizontal scaling shows its true potential when having heavy concurrent traffic. For example, the system with four detection engines at 480 concurrent requests takes as long as the single system takes for 120 or the doubled system for 240. The slightly higher response time average of the scaled framework with four detection engine instances probably originates from the e-mail server stressing to handle 480 connections at a time. Delay caused by the load balancer is negligible and thus can be excluded as a cause.

From Fig. 13, it is directly recognizable that scaling increases the e-mail throughput. Again, a linear relationship between the results for each system is recognizable. However, the shapes of the lines reveal an increasing slope from 20 to 240 concurrent connections, but a decrease in growth rate from 240 onwards. Having eliminated the detection engine bottleneck through scaling, this implies that the e-mail server is approaching its limits and has already a restricting impact at 480 concurrent mails. Problems with reaching the e-mail server's limits can be easily alleviated through placing another load balancer in front of it and multiplying mail server instances.

For an extreme use case, a company might handle 20 million e-mails a day, which are 230 mails per second [55]. A single Haraka instance can handle more than 45 mails per second without extraction; with extracting images, 19 mails per second were measured. To provide a buffer, we assume a throughput of 15

mails per second. Then it would require just 16 e-mail server containers to handle the full traffic of that company. Assuming further linear behavior, it requires 10 detection engine instances per Haraka instance. In total it would require 16 e-mail server instances with 160 detection engine instances to filter and route the complete e-mail traffic of the company at reasonable speed. Slightly over-scaling provides enough buffers for peak traffic, but still the service would just need more time but not throw errors when being overloaded. Having a 1:10 ratio of mail server to detection engines recommends future improvement of the detection engine's performance. Also, the 16 Haraka instances might be scaled down through separating the mail extractor from the server to a dedicated service.

Regarding the replacement of links, the regular expression method is commonly seen as one of the fastest methods. However, it can be discussed if the replacing should be executed on the server or rather also be handled via its own micro-service. These two measures could cut the number of required mail server instances by half.

Evolving to a full micro-service system would be beneficial for scalability but would also add complexity on the service level. Then, HTTP may be replaced by using web sockets since communication frequency would increase and this switch of protocols could reduce overhead. This approximation shows that the framework is prepared for extremely large-scale enterprise usage, although the detection engine would require improvements to reduce the required number of instances.

Finally, it can be concluded that all defined objectives and technical requirements are fulfilled. Especially the limits of maximum five minutes e-mail delay, handling more than 20 concurrent connections with less than 250 MB memory usage, was exceedingly satisfied.

5.4. Static code quality

Code quality usually does not affect the execution of the program but does greatly affect its further development. Code quality analysis was executed through using the JetBrains *Webstorm* JavaScript IDE and the *ESLint* plugin. Results of the static code analysis indicate a high code quality level and application of conventions and best practices as well as utilization of a proper code style. This high standard is also expected by the open source community in order to attract contributors for further development.

5.5. Evaluation against objectives

As part of the Design Science Research Method, this section evaluated the solution against the defined objectives. This section integrates the different aspects in an overall assessment. Table 4 lists the requirements as well as their evaluation in the software prototype. Concerning the first requirement, the design solution fulfills the approach of selective blocking by implementing an identify and block strategy for tracking images. Concerning the detection algorithm (requirements 2 and 3), the software solution relies on previously developed and tested algorithms for detection [12]. However, given the modular architecture of the solution, the detection engine can be easily updated or exchanged, which fulfills the fourth requirement. Concerning the performance and scalability of the solution (requirement 5), different tests were reported before that reveal the solution to fulfill typical enterprise-related performance requirements. Finally, the solution is realized using a central approach at the incoming mail server (central solution) which makes the detection and prevention independent from the individual endpoint to cover all potential mail access clients (requirement 6).

In summary, the solution fulfills all the requirements set to an appropriate extent. The software is suitable to protect e-mail recipients against current image tracking approaches in an enterprise context.

6. Discussion

Previous empirical findings, for example that around 92% of all e-mail openings of commercial newsletters might be unprotected against tracking [15], indicate the scope of the privacy problem and motivate our research. Concerning the differentiation to other existing protection services against e-mail tracking, there are a few other services that broadly address the same issue, but always with problems; either dependency on a specific browser or a very complicated procedure to setup the service. Summarizing, earlier privacy solutions either lack user friendliness, precision or independency.

In general, it should be questioned why the user should be responsible for making his own e-mail inbox tracking free, and why the service provider is not offering better privacy protection. Furthermore, companies might want to protect their employees' work e-mail addresses from tracking. A simple browser or inbox plugin could also be attractive to end-users, but it would always come along with a dependency on a third-party application. Our goal was to remain independent and let many users benefit if a mail service implements our framework.

The motivation of this study was to create an enterprise-grade framework that fills the identified gap in the field of privacy-enhancing technologies. Recalling the evaluation section, it can be clearly said that the framework fulfills these criteria. The framework is ready and suitable to be applied in actual production conditions. Metrics from the evaluation sections show that this is achieved through right choices regarding technologies and a solid software design. The ability to handle more than 400 concurrent connections with reasonable response times on low resources and with a 0% error rate proves the framework's quality.

However, meeting an end-user demand is not the only purpose of this software. It was also built to provide a basis for further research in the field of e-mail tracking. The platform aims to be used by researchers to test new detection engines, gather data about the problem itself, and to serve as a real-world test environment. Additionally, the selective filtering approach was demonstrated as valid through implementation in a real application. This framework augments the research field by a practical and extensible system to support investigations of any direction.

Concerning limitations, there are aspects that would also have supported the decision in favor of monolithic application architecture. However, monolithic designs are known for their struggles with scaling. Although the architecture would provide an environment for easier logging and features such as shared memory, these advantages do not outweigh the benefits of a micro-service architecture. As tracking being a rapid evolving domain, the flexibility of micro-services allows exchanging parts without the need to redesign the whole solution, which fosters further development.

A similar, but contrary argument could go even further in the direction of micro-services. A further decoupling of the mail extraction and the replacing would free resources on the mail server. On the other hand, this would lead to higher communication traffic in the local network and thus would the HTTP overhead be higher. Another point against it is that there are fixed resources required by every container such as a runtime environment and libraries as well as a new load balancer if multiplication is planned. For upcoming refactoring iterations, decoupling of both operations into separate services may be considered.

For a detection engine that has to handle heavy traffic, needs to process HTTP, and should run on limited resources, R might be

Table 4
Requirement-based software evaluation.

ID	Requirement	Category	Realization/Evaluation
1	Selective blocking	Concept	Selective blocking is realized by the solution.
2	Accuracy	Algorithm	Very accurate detection algorithms are used in the detection engine (Section 4.6; [12]).
3	Universality	Algorithm	Detection engine can classify unknown e-mails (Section 4.6), from different senders and at a different time [12].
4	Modularity	Architecture	Modular software architecture allows to exchange detection components.
5	Scalability	Architecture	Software tests demonstrated reasonable performance and scalability of the system.
6	Coverage of heterogeneous endpoints	Architecture	Central e-mail server (proxy) allows for endpoint coverage irrespectively of e-mail client.

a suboptimal choice. Consequently, future detection engine development could further improve execution speed, complexities, and resource usage. In the field of machine learning, there are alternatives such as Python or Julia, which are also popular and proven in large-scale applications.

Yet another aspect concerns the discussion if JavaScript should be used in use cases such as e-mail. The community is divided in that point. If technologies emerge that offer better advantages than Node.js, then a mail server re-implementation could be recommended. For now, it was considered the most efficient framework to develop the artifact.

Concerning future research, the implementation of the detection engine could be improved in order to even better meet the identified requirements of the framework. It would be interesting to know if there is a more efficient method than the currently used one or even a possibility to split the detection engine into different tasks with certain checkpoints. There could also be an extension of the detection engine and the mail extractor to not only check for tracking images, but tracking links in general. Though links involve fewer risks, users may be clicking on tracking links by accident.

A further next step would be the enterprise-grade application of the framework. Gathering data about the framework's behavior in a non-experimental setting would contribute to future improvements. Real usage data has also a positive effect on adoption by the open source community.

An extension that would foster future privacy research would be the implementation of a proper logging engine for tracking attempts and a sophisticated analytics tool. These two features would be of high value for quantitative research in e-mail tracking. Metrics such as the tracking images' domains and other characteristics would become analyzable. Also, the resulting database from these two services would be beneficial to the general e-mail research community.

To foster application and future research, we publish the source code of our framework in an open repository *GitHub* under an open source license [13]. Together with this article, this supports *communication*, the final step in the formal process of design science research [14].

7. Conclusion

Tracking methods, such as web and e-mail tracking, are popular marketing tools. Data derived from tracking provides valuable information regarding a person's interests and reception behavior. Modern e-mail tracking methods allow the sender to determine how often an e-mail was opened, the device used to read the e-mail, which links were clicked, and the location and time when the recipient opened an e-mail. While different aspects concerning e-mail tracking have been studied in isolation, a study integrating former efforts in a ready to use countermeasure is still missing. This contribution addresses this gap, by conceptualizing and implementing a novel protection framework against e-mail tracking.

Following the Design Science Research Method, we developed a software being capable to identify tracking images in e-mails via machine learning with very high accuracy and can selectively replace them so that an untracked e-mail is provided for the end user without any manual effort. Our mail protection framework is an enterprise-grade software, flexibly extensible, highly scalable, and ready to be applied in actual production conditions. The experimental evaluation section shows that this is achieved through corresponding choices regarding technologies and the creation of a solid and flexible software design.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] A. Goldfarb, C. Tucker, Privacy and innovation, Innov. Policy Econ. 12 (1) (2012) 65–90.
[2] T. Ermakova, A. Hohensee, I. Orlamünde, B. Fabian, Privacy-invading mechanisms in E-commerce – A case study on german tourism websites, Int. J. Netw. Virtual Organ. 20 (2) (2019).
[3] T. Ridley-Siegert, DMA insight: consumer email tracking report 2015, J. Direct Data Digit. Mark. Pract. 17 (3) (2016) 163–169.
[4] S. Englehardt, J. Han, A. Narayanan, I never signed up for this! Privacy implications of email tracking, Proc. Priv. Enhanc. Technol. 2018 (1) (2018) 109–126.
[5] B. Fabian, B. Bender, L. Weimann, E-mail tracking in online marketing - Methods, detection, and usage, in: Paper Presented at the 12. Internationale Tagung Wirtschaftsinformatik (Wirtschaftsinformatik 2015), Osnabrück, 2015.
[6] T.-C. Li, H. Hang, M. Faloutsos, P. Efstathopoulos, TrackAdvisor: Taking back browsing privacy from third-party trackers, in: J. Mirkovic, Y. Liu (Eds.), Passive and Active Measurement, vol. 8995, Springer International Publishing, 2015, pp. 277–289.
[7] L. Vaas, M. Stockley, How emails can be used to track your location and how to stop it, 2014, Retrieved from https://nakedsecurity.sophos.com/2014/02/27/how-emails-can-be-used-to-track-your-location-and-how-to-stop-it/.
[8] S. Yu, Crime hidden in email spam, in: Encyclopedia of Criminal Activities and the Deep Web, IGI Global, 2020, pp. 851–863.
[9] A. Dabrowski, G. Merzdovnik, J. Ullrich, G. Sendera, E. Weippl, Measuring cookies and web privacy in a post-GDPR world, in: Paper Presented at the Passive and Active Measurement, Cham, 2019.
[10] B. de Roos, The Implementation of Privacy Regulation: Predictors of the Discrepancy Between Attitude and Behavior of Organizations (Master thesis), Open Universiteit, Faculteit Management, Science & Technology, 2020, Retrieved from https://research.ou.nl/ws/portalfiles/portal/30774606/Roos_de_B_IM9806_BPMIT_scriptie_Pure.pdf.
[11] B. Bender, B. Fabian, J. Haupt, S. Lessmann, T. Neumann, C. Thim, Track and treat—Usage of e-mail tracking for newsletter individualization, in: Paper Presented at the Proceedings of the 26th European Conference on Information Systems (ECIS), Portsmouth, UK, 2018.

[12] J. Haupt, B. Bender, B. Fabian, S. Lessmann, Robust identification of email tracking: A machine learning approach, European J. Oper. Res. 271 (1) (2018) 341–356.
[13] B. Hesseldieck, B. Fabian, B. Bender, J. Haupt, S. Lessmann, Enterprise Email Tracking Prevention Suite. GitHub Repository, 2020, Retrieved from https://github.com/ben-fabian/enterprise_email_tracking_prevention.
[14] K. Peffers, T. Tuunanen, M.A. Rothenberger, S. Chatterjee, A design science research methodology for information systems research, J. Manage. Inf. Syst. 24 (3) (2007) 45–77.
[15] B. Bender, B. Fabian, J. Haupt, S. Lessmann, E-mail tracking: Status quo and novel countermeasures, in: Paper Presented at the Proceedings of the 37th International Conference on Information Systems (ICIS), Dublin, Ireland, 2016.
[16] Campaign Monitor, How do I create a printer-friendly email newsletter?, 2010, Retrieved from https://www.campaignmonitor.com/blog/post/3232/how-do-i-create-a-printer-friendly-email-newsletter/.
[17] M. Agosti, G.M. Di Nunzio, Web Log Mining: A study of user sessions, in: Paper presented at the 10th DELOS Thematic Workshop on Personalized Access, Profile Management, and Context Awareness in Digital Libraries (PersDL 2007), 2007.
[18] T. Ermakova, B. Fabian, B. Bender, K. Klimek, Web tracking – A literature review on the state of research, in: Paper Presented at the The Hawaii International Conference on System Sciences (HICSS 51), 2018.
[19] A. Javed, POSTER: A footprint of third-party tracking on mobile web, in: Paper Presented at the Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, 2013, http://dl.acm.org/citation.cfm?doid=2508859.2512521.
[20] C. Jensen, C. Sarkar, C. Jensen, C. Potts, Tracking website data-collection and privacy practices with the iWatch web crawler, in: Paper Presented at the Proceedings of the 3rd Symposium on Usable Privacy and Security, 2007.
[21] S. Mittal, User Privacy and the Evolution of Third-Party Tracking Mechanisms on the World Wide Web, Stanford University, 2010, Retrieved from https://purl.stanford.edu/hw648fn9717.
[22] J. Parra-Arnau, Pay-per-tracking: A collaborative masking model for web browsing, Inf. Sci. 385–386 (2017) 96–124.
[23] A. Alsaid, D. Martin, Detecting web bugs with bugnosis: Privacy advocacy through education, in: G. Goos, J. Hartmanis, J. van Leeuwen, R. Dingledine, P. Syverson (Eds.), Privacy Enhancing Technologies, vol. 2482, Springer, Berlin Heidelberg, 2003, pp. 13–26.
[24] F. Fonseca, R. Pinto, W. Meira, Increasing User's Privacy Control Through Flexible Web Bug Detection, 2005, http://ieeexplore.ieee.org/document/1592379/.
[25] D. Martin, H. Wu, A. Alsaid, Hidden surveillance by Web sites: Web bugs in contemporary use, Commun. ACM 46 (12) (2003) 258.
[26] A. Yamada, M. Hara, Y. Miyake, Web tracking site detection based on temporal link analysis and automatic blacklist generation, J. Inf. Process. 19 (2011) 62–73.
[27] T. Bujlow, V. Carela-Espanol, J. Sole-Pareta, P. Barlet-Ros, A survey on web tracking: Mechanisms, implications, and defenses, in: Proceedings of the IEEE, 2017, pp. 1–35.
[28] P. Leon, B. Ur, R. Balebako, L. Cranor, R. Shay, Y. Wang, Why Johnny can't opt out: a usability evaluation of tools to limit online behavioral advertising, in: Paper Presented at the Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 2012.
[29] I. Sanchez-Rola, X. Ugarte-Pedrero, I. Santos, P.G. Bringas, The web is watching you: A comprehensive review of web-tracking techniques and countermeasures, Log. J. IGPL 25 (1) (2017) 18–29.
[30] A. Bouguettaya, M.Y. Eltoweissy, Privacy on the web: facts, challenges, and solutions, IEEE Secur. Privacy Mag. 1 (6) (2003) 40–49.
[31] W.T. Harding, A.J. Reed, R.L. Gray, Cookies and web bugs: What they are and how they work together, Inf. Syst. Manag. 18 (3) (2001) 17–24.
[32] A. Bilos, D. Turkalj, I. Kelic, Open-rate controlled experiment in E-mail marketing campaigns, Trziste 28 (1) (2016) 93–109.
[33] M. Hartemo, Email marketing in the era of the empowered consumer, J. Res. Interact. Mark. 10 (3) (2016) 212–230.
[34] X. Luo, R. Nadanasabapathy, A.N. Zincir-Heywood, K. Gallant, J. Peduruge, Predictive analysis on tracking emails for targeted marketing, in: N. Japkowicz, S. Matwin (Eds.), Discovery Science, vol. 9356, Springer International Publishing, 2015, pp. 116–130.
[35] X. Zhang, V. Kumar, K. Cosguner, Dynamically managing a profitable email marketing program, J. Mark. Res. (2017).
[36] A. Bonfrer, X. Drèze, Real-time evaluation of e-mail campaign performance, Mark. Sci. 28 (2) (2009) 251–263.
[37] A.B.I. Hasouneh, M.A. Alqeed, Measuring the effectiveness of e-mail direct marketing in building customer relationship, Int. J. Mark. Stud. 2 (1) (2010).
[38] M. Bhattacharyya, S. Hershkop, E. Eskin, MET: An experimental system for malicious email tracking, in: Paper Presented at the Proceedings of the 2002 Workshop on New Security Paradigms, 2002.
[39] S.J. Stolfo, S. Hershkop, K. Wang, O. Nimeskern, C.-W. Hu, A behavior-based approach to securing email systems, in: G. Goos, J. Hartmanis, J. van Leeuwen, V. Gorodetsky, L. Popyack, V. Skormin (Eds.), Springer Berlin Heidelberg, 2003, pp. 57–81.
[40] G.A. Grimes, M.G. Hough, M.L. Signorella, Email end users and spam: relations of gender and age group to attitudes and actions, Comput. Hum. Behav. 23 (1) (2007) 318–332.
[41] S. Hameed, T. Kloht, X. Fu, Identity based email sender authentication for spam mitigation, in: Paper Presented at the Eighth International Conference on Digital Information Management (ICDIM 2013), 2013, http://ieeexplore.ieee.org/document/6694015/.
[42] A. Herzberg, DNS-Based email sender authentication mechanisms: A critical review, Comput. Secur. 28 (8) (2009) 731–742.
[43] X. Gu, M. Yang, C. Shi, Z. Ling, J. Luo, A novel attack to track users based on the behavior patterns, Concurr. Comput.: Pract. Exper. 29 (6) (2017).
[44] J. Sammons, M. Cross, Email safety and security, in: The Basics of Cyber Safety, Elsevier, 2017, pp. 75–86.
[45] N. Tsalis, A. Mylonas, D. Gritzalis, An intensive analysis of security and privacy browser add-ons, in: C. Lambrinoudakis, A. Gabillon (Eds.), Risks and Security of Internet and Systems, vol. 9572, Springer International Publishing, 2016, pp. 258–273.
[46] J. Martin, Rapid Application Development, Macmillan Pub. Co.; Collier Macmillan Canada; Maxwell Macmillan International, 1991.
[47] M.L. Despa, Comparative study on software development methodologies, Database Syst. J. 5 (3) (2014) 37–56.
[48] N. Alshuqayran, N. Ali, R. Evans, A systematic mapping study in microservice architecture, in: Paper Presented at the 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA), 2016.
[49] J. Cook, The docker engine, in: Docker for Data Science, Apress, 2017, pp. 71–79.
[50] P. Krill, Why R? The pros and cons of the R language, 2015, Retrieved from https://www.infoworld.com/article/2940864/application-development/r-programming-language-statistical-data-analysis.html.
[51] D. Vohra, Docker services, in: Docker Management Design Patterns, Apress, 2017, pp. 55–84.
[52] P. Arijs, Docker usage statistics: Increased adoption by enterprises and for production use, 2016, Retrieved from https://www.coscale.com/blog/docker-usage-statistics-increased-adoption-by-enterprises-and-for-production-use.
[53] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning: Prediction, Inference and Data Mining, Springer-Verlag, New York, 2009.
[54] R. Braden, Requirements for internet hosts-communication layers (2070-1721), 1989, Retrieved from https://tools.ietf.org/html/rfc1122.
[55] M. Sergeant, Large scale haraka users, 2018, Retrieved from https://haraka.github.io/users.html.