Spamato – An Extendable Spam Filter System*

Keno Albrecht, Nicolas Burri, Roger Wattenhofer
Computer Engineering and Networks Laboratory
ETH Zurich, 8092 Zurich, Switzerland
{kenoa, burri, wattenhofer}@tik.ee.ethz.ch

Abstract

Spam filter developers are confronted with the task of integrating their ideas in user-friendly products. In this paper, we introduce Spamato as an open, extendable, and multi-faceted spam filter framework. Spamato provides fundamental services commonly required by filter developers to facilitate the implementation of new approaches. Furthermore, we support email clients with add-ons to enable users to intuitively collaborate with Spamato. We also present a variety of filters and exhibit an evaluation of URL-based techniques.

1 Introduction

Spam, phishing, and virus infected messages are indubitably some of the primary annoyances of the Internet experience. According to MessageLabs, in 2004 nearly three out of four inspected messages have been classified as unsolicited [1]. Although some organizations argue that the peak of the spam vexation has passed, others like Spamhaus predict an unabated increase to a spam rate of 95 percent by mid-2006 [2].

Several approaches on how to deal with spam have been discussed. The proposals are manifold and contain suggestions like legal regulations [3, 4], economic burdens [5, 6], DNS-based attempts [7, 8], and server- and client-side solutions using a variety of filtering techniques.

While some governments around the world slowly begin to enact and enforce laws punishing spammers, legal proposals for the Internet suffer from national limitations. Even if mass spammers like Jeremy Jaynes are sentenced to 9 years of prison in the United States [9], similar businesses in China or Russia are not considered illegal.

Also technical approaches like increasing the costs of bulk mailing or the addition of security features to the mail exchange protocol might take years before becoming reality. Differing interests prevent the fast development of a new standard protocol to replace SMTP which would be necessary to realize many of the proposed spam countermeasures. Unilateral approaches like the Sender ID Framework (SIDF) [8] from Microsoft or the DomainKeys system [7] from Yahoo may reduce the amount of spam to a certain extent but for a significant improvement of the situation a global solution is required.

We believe that no panacea exists to remedy the spam problem; even combinations of the proposed solutions will not eliminate spam in the near future. If we assume that spammers will always be able to advertise their products by email, the primary remaining question is how to prevent people from reading these messages.

Spam filters do not aim to avoid spam but to ease the task of manually identifying and separating spam from "ham" (wanted) messages, saving user time and company money. Spam filters can be employed on the server-side, either at an ISP or in a corporation, or on the client-side, being part of a user's email client.

While there are dozens of different spam filters available, most of them have been built independently of each other. Every author of a spam filter must reinvent the wheel; many fundamental operations such as accessing and parsing a message have repeatedly been implemented for every new filter. Also the integration of filters into an email client and their final deployment are important to reach a large user community. As the installation of several spam filtering tools on one machine often leads to undesired side effects the users have to decide on one system leading to unnecessary competition amongst the filtering tools. To overcome the redundant work and to simplify the development of new filters, we introduce *Spamato* as an open and extendable spam filter framework.

Spamato aims to bring a practical, easy-to-use, and effective spam filter technology to the user's desktop. It has been designed to be used primarily as an email client add-on, allowing users to control and adjust the filtering process from within their email client. By providing collaboration support and an intuitive graphical interface, Spamato involves users in the spam decision task and benefits from their feedback. Furthermore, the combination of multiple filtering techniques leads to a high spam detection rate and a low false-positive rate.

On the technical site, Spamato is an extendable framework. It consists of a *Core* component providing basic services—such as communication, event handling, and

* The work presented in this paper was supported (in part) by the Hasler Stiftung under grant number 1828.

security facilities—and plug-ins extending the Core. The framework has been written in Java and can be used on all major platforms making it accessible for many developers. The Spamato Core offers interfaces to write email client add-ons as well as new filters and other extensions.[1] We want to encourage spam filter developers to rely on Spamato as a proven and tested framework instead of bothering with redundant work. Besides the basic functionalities Spamato also contains a statistics engine which helps to evaluate new filters and to compare their efficiency to other available algorithms.

This paper introduces Spamato; we explain the concept of Spamato, present details about the filters we have implemented on top of the Spamato framework, and highlight the system's benefits for users and developers. The remainder of this paper is organized as follows: In the next chapter, we present work related to ours. The third chapter gives a technical overview of the Spamato framework detailing the add-on concept, the plug-in mechanism, and the filtering process. In Chapter 4, we describe and evaluate our filters. Finally, we conclude the paper in Chapter 5.

2 Related Work

Existing spam filter systems can be classified as follows: Server side filter systems are maintained by trained administrators. In contrast, client side filtering systems are designed to work without professional administration on top of an email client or to be used as a proxy between the email client and the email server. In this section, we present a short overview over the most prominent representatives of both classes.

On the server side *Procmail* [10] is the tool commonly used to access emails. It directly manipulates messages in the mbox file written by the server and allows external plug-ins to decide how to process the messages. The main drawbacks of the Procmail system are its complicated setup and maintenance. Only trained administrators are able to configure a whole Procmail system without the risk of breaking the mail system. The most well known spam filtering tool running on top of Procmail is the Apache *SpamAssassin* [11, 12]. SpamAssassin uses an extendable rule system to classify email messages. Its popularity is based on the large number of available filtering algorithms, which can be used as a SpamAssassin rule. Additional rules implemented in Perl can be added to customize the behavior of the system. The downside of the system is its complicated setup and the only rudimentary available user feedback channel. To report a missed spam message a command line call needs to be executed. Consequently, additional software is required to report spam mails if the user does not have direct access to the SpamAssassin installation. This functional deficit is especially problematic for a filter rule like *Vipul's Razor* [13] which identifies spam mails by comparing incoming messages to a centralized spam database which collects manual spam reports from all users of the system.

SpamGuru [14] is another server side filtering system developed by IBM. Unlike SpamAssassin and Spamato, SpamGuru is a closed source project and thus not extendable by external developers. SpamGuru uses an optimized ordering of filter mechanisms to maximize the message throughput of a server. A plug-in for the Lotus Notes mail client provides a convenient user feedback channel that is used to report spam messages to a collaborative spam filter which is part of the SpamGuru system. Clients other than Lotus Notes are currently not supported.

On the client side numerous email filters are available. Unfortunately, many of these tools just use one single filtering algorithm which limits their effectiveness. Beside these single filter tools, there are also some filtering suits available which combine several different algorithms.

Cloudmark's SafetyBar [15] (formerly SpamNet) is a commercial Microsoft Outlook(-Express) add-on. It employs several filtering techniques to identify messages but mostly relies on collaborative filters. The SafetyBar add-on is an extended version of Vipul's Razor and thus accesses the same database as the Razor system. Cloudmark has not released the source code of their SafetyBar making it impossible to extend the system with other filters. Up to now, mail clients other than Outlook(-Express) are not supported.

SpamPal [16] is a client side filtering suite which is designed to be email client independent. Instead of using a special email client add-on, SpamPal acts as a transparent proxy between the email client and the email server. SpamPal is an open source project and supports custom extensions which have to be implemented in C. The system can only be used on Microsoft Windows systems and does not provide a user feedback channel making it impossible to employ collaborative filters.

3 System Overview

In this chapter, we first describe how the Spamato system architecture has been designed. Then, we highlight the add-on scheme, explain the dynamic plug-in mechanism and, finally, detail how the system processes emails.

3.1 System Architecture

Figure 1 illustrates the Spamato system architecture and its main components. The *Spamato Core* provides key services to dynamically loaded plug-ins and extensions of the Core. The *Spamato Factory* and the *Plug-in Container* are the only parts of the Core that are statically linked to it. They cannot be omitted since they launch

[1] Currently, add-ons for Microsoft Outlook and the Mozilla Mail Client exist; Thunderbird will be supported soon. To support other mail clients, a stand-alone email server proxy is also available. Five different filters have been implemented so far.

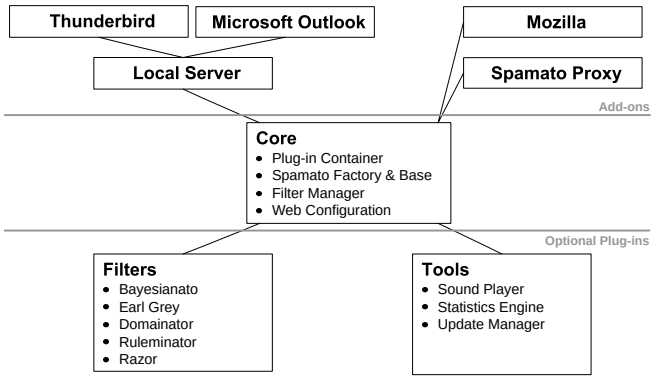

Figure 1: Spamato consists of the Spamato Core which can be extended by plug-ins and email client add-ons.

Spamato's elementary services and initialize the system and the plug-ins.

Spamato has been built according to the "everything is a plug-in" paradigm. In fact, even some of Spamato's primary features are bundled in a mandatory plug-in, the *Spamato Base*. Also the *Filter Manager* and the *Web Configuration* are essential to the whole system. While the first is a mere repository for all filters, the latter embeds a simple HTTP server and shares its service with other plug-ins (the plug-in mechanism is presented in Section 3.3). Thereby, plug-ins can be configured using a common web browser.

In contrast to compulsory plug-ins, the *Sound Player*, which enables the Spamato Base to play a short jingle when a spam message has been detected, as well as the *Statistics Engine*, which sends data about the filtering process to a server, are optional and can be omitted without harming the system. Again, using the Web Configuration the Sound Player can be configured to play custom jingles and the behavior of the Statistics Engine can be adjusted or completely turned off. Of course, arbitrary filters can be added or removed from Spamato and are (obviously) optional too. (In Chapter 4, we describe the current filters in detail.)

3.2 Email Client Add-Ons

Spamato provides an interface to facilitate the development of email client add-ons. Such add-ons can contain buttons and other graphical widgets for representing information and user feedback. Displaying information to users in their email client, such as the number of detected spam messages, is an important feature used to involve the user in the system. Receiving user feedback is a fundamental requirement to support learning and collaborative filters. Otherwise, learning filters like a Bayesian-based filter can hardly develop their capability to distinguish between spam and ham. Literally, a collaborating filter cardinally relies on users' collaborative rating to achieve a significant decision. Such a rating can only be computed if users are able to vote for spam or ham; this procedure is most effectively supported by adding meaningful "report" (spam) and "revoke" (ham) buttons to the user's email client.

Unfortunately, the implementation of add-ons is not standardized. Hence, every email client has to be supported in a different way. For instance, our Spamato add-on for Microsoft Outlook is written in Visual Basic/C#, while the add-ons for Mozilla and Thunderbird are implemented using a combination of the XML User Interface Language (XUL) and JavaScript. Another important point in add-on development is whether the email client is capable of directly supporting Java. If so, the Spamato framework can be accessed by the add-on without further consideration (such as Mozilla). Otherwise, Spamato has to be invoked by a *mediator* in between (see *Local Server* in Figure 1); this is the case for Outlook and Thunderbird. These add-ons communicate with the mediator using an XML-based scheme while, subsequently, the mediator translates the requests into normal Java method calls and vice versa. Apparently, the second approach is more complex, but the burdens are constituted by the developers of the email clients. Spamato eases the implementation of add-ons by providing the Local Server.

3.2.1 The Spamato Proxy

We also provide the *Spamato Proxy* in order to use Spamato with email clients which are not supported by an add-on yet. The Proxy works similarly as the aforementioned mediator between an email client and the Spamato system. Additionally, it also relays emails between an email client and a user's normal email server acting as a transparent email server proxy (hence its name) to the client. In contrast to the mediator, since no add-on exists, the Proxy can neither be controlled via buttons for the purpose of gaining feedback for the system nor can it directly receive messages from the email client by a Java method call or in the XML format. Instead, the Proxy intercepts each message sent from the email server and checks if it is spam before forwarding it to the email client.

More precisely, the IMAP service of the Spamato Proxy works as follows. All requests from the email client are first sent to the Proxy. Usually, the requests and their replies are transparently tunnelled to the real server and back to the client without intervention. Only *Fetch* and *Copy* commands have to be handled separately: Fetch commands, which are sent to request a message or its headers, are intercepted, to screen whether it is spam or ham. If the message is considered innocent it is delivered to the client. Otherwise, the message is moved to a special spam folder on the IMAP server. Copy commands that copy messages from or to the special spam folder are used to indicate report or revoke attempts. If a user wants to report an unrecognized spam message to the Spamato system, the message can be dropped to the special spam folder. On the other hand, the activity of

Listing 1 The Spamato Base `plugin.xml` File

```
<plugin>
<name>Spamato</name>
<description>The Spamato Base</description>
<class>spamato.common.main.SpamatoImpl</class>
<version>0.2</version>
<update-url>
 http://spamato.ethz.ch/update
</update-url>
<requires><permission type="all"/></requires>
<share>
 <package name="spamato.common"/>
 <package name="com.thoughtworks.xstream"/>
 <extension-point id="spamato.filters"/>
</share>
</plugin>
```

Listing 2 The Earl Grey `plugin.xml` File

```
<requires>
 <permission type="all"/>
 <plugin key="spamato">
 <extension point="spamato.filters"
class="...EarlGreyClient/>
 </plugin>
 <plugin key="web_config" name="Configuration">
 <extension point="config.web.pages"
handler="...EarlGreyPageHandler" menu="Earl Grey
Filter"/>
 </plugin>
</requires>
```

moving a message from the spam folder to the Inbox is regarded as a revoke of this message.

The Proxy also works with POP3 accounts. In this case, the Proxy adds a header stating the result of the spam check before forwarding it to the email client. Subsequently, the email can be handled by an email client's built in filtering facility, for example by moving it to a special folder or by deleting it immediately. Although this approach is less sophisticated than our IMAP solution, it works fine with almost any email client and server.

On POP3 accounts, the Proxy also allows for user feedback by providing an SMTP service. It intercepts forwarded messages from the client sent to a special local email address either as a spam report or a ham revoke. Unfortunately, email clients forward messages in different ways which makes it difficult to build a single solution to process them.

Although the Proxy approach is more limited in offering its services to the user, the configuration of Spamato and its plug-ins is no problem. The Web Configuration is operated with a common browser, and does not depend on the user's email client.

3.3 Plug-in Mechanism

As stated before, plug-ins are basic building blocks in the Spamato system. Plug-ins can either be mandatory, such as the Spamato Base or the Web Configuration, or optional, like the Sound Player or filters. Plug-ins can provide their services to other plug-ins. Additionally, they can communicate with each other using a publish/subscribe event mechanism. In this section, we present some of these aspects.

3.3.1 The `plugin.xml` File

The extract of the `plugin.xml` shown in Listing 1 describes the Spamato Base plug-in. The `<name>` of the plug-in and the informal `<description>` are used solely to describe the plug-in and its purpose. The main `<class>` is initiated when this plug-in is loaded. The `<version>` and the `<update-url>` provide information to the plug-in mechanism and to the Update Manager plug-in (if available) to initiate the default plug-ins and, possibly, to obtain newer versions.

The `<requires>` section of the XML file specifies requirements on the Spamato framework or other plug-ins. In this example, the Spamato Base asks the framework for `"all"` permissions. This allows the Base, for example, to read from and write to the local hard disk as well as to connect to arbitrary Internet servers; more restrictive rules are possible. The `<requires>` section can also contain entries to subscribe for particular events—for example, the Sound Player subscribes to the `mail.post_check` event that is published after the Spamato system has made its final spam decision about a message—or, as we show later in Listing 2, to hook into a shared extension point.

The `<share>` part enables other plug-ins to extend or use the facilities provided by the sharing plug-in. The Spamato Base allows other plug-ins to access the package `spamato.common` that contains common utility classes and the XStream [17] package `com.thoughtworks.xstream` which is used to exchange data among different client and server components. Furthermore, the Spamato Base offers the *extension point* `spamato.filters` which is necessary to register filters.

Listing 2 depicts how the Earl Grey filter (see Section 4.1.2 for more details on this filter) registers with this extension point. In the `<requires>` section of the `plugin.xml` file, the "`spamato`" plug-in is extended by specifying the `class` that has to be accessed for the `spamato.filters` extension point. Additionally, the interaction with the Web Configuration is shown. To give users the ability to adjust the filter's parameters, the filter has to hook into the `config.web.pages` extension point by registering a handler that creates a configurable HTML page. This page is displayed when the user selects the corresponding `menu` entry that is automatically created by the Web Configuration. Apparently, extension points can define arbitrary parameters and thus obtain all the information necessary to fulfil their tasks.

3.3.2 Loading Plug-ins

Technically, a plug-in has to implement the `Plug-in` interface, which is located in the Spamato Core. In addition, it has to provide a `plugin.xml` file (see Section 3.3.1) and it must be placed in the plug-ins directory of the Spamato system either in the form of several class files or as a single compressed `plugin.zip` file. A plug-in that meets these requirements is automatically loaded by Spamato when it is started or reinitiated, for instance by the Update Manager after a new version of a plug-in has been downloaded.

The `plugin.xml` file characterizes the interaction with other plug-ins. It has to be parsed in order to add the plug-in to the Plug-in Container and to arrange its dependencies. It is beyond the scope of this paper to detail all aspects. Still we want to point out the role of graph-like organized Java ClassLoaders to achieve the sharing facility of packages and the usage of extension points. Each plug-in is loaded and instantiated using its private ClassLoader. Plug-ins sharing packages enable other plug-ins to use these "free" classes. On the other hand, hooking into an extension point entails the plug-in offering the extension point to call methods of the extending plug-in. In both cases, one plug-in has to access the ClassLoader of another plug-in.

We highlight this point not for mere technical reasons but to emphasize the default state: By default, no plug-in can use or even knows about classes of other plug-ins; they are totally shielded in their personal namespaces. This results in three features. First, developers do not have to worry about other plug-ins. They can label their packages without considering problems due to overlapping namespaces although all plug-ins are dynamically loaded into the same JVM. More precisely, developers can even prohibit the access to their classes. Second, the testing and analysis of filters, which are themselves plug-ins, are facilitated. The same filter can be used multiple times in one Spamato instance with different settings just by copying it into different directories in the plug-ins directory. Thus, it is possible to easily compare different settings and to improve the filter's success rate. Finally, using separate ClassLoaders provides the capability to update plug-ins without the need for restarting the whole Spamato system.

3.4 Filtering Process

Spamato is designed to support several filters simultaneously. The report and revoke procedures are rather straightforward implementations; all filters are sequentially notified of the operation and process the message if necessary. In this section, we describe the task of identifying spam messages, which demands a more elaborate approach.

During the filtering process, each filter contributes to the final evaluation, spam or ham. Figure 2 illustrates how messages are processed to obtain this decision. Generally, the procedure is triggered if a new message arrives. Each filter then has the chance to *pre-check* the message in order to denote if the message has to be filtered at all. Subsequently, the real *checks* are performed and their results accumulated to calculate the overall spam probability. Finally, this result is returned to the user, and the filters can adapt to the decision in the *post-check* stage. We now describe the five phases in more detail.

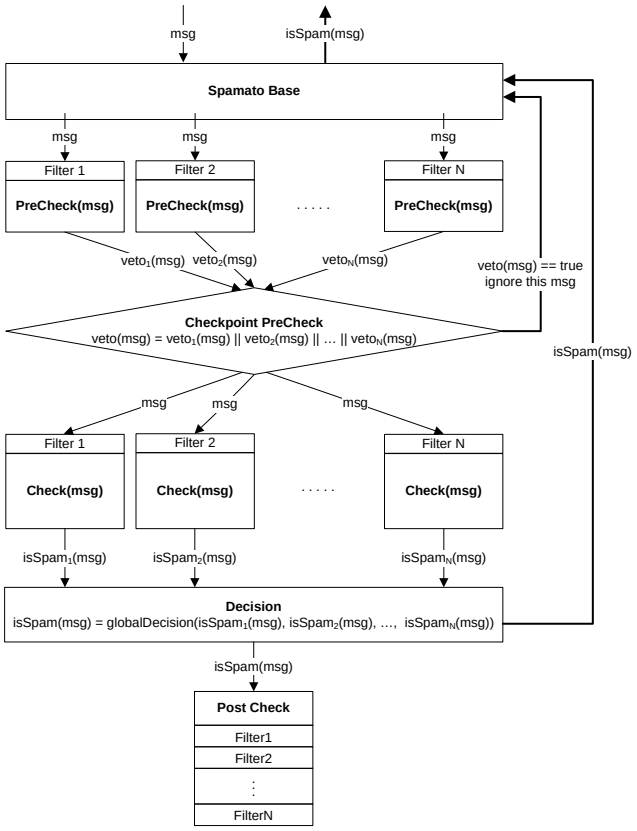

Figure 2: The filtering process consists of five phases. The overall spam probability of a message is based on the evaluation of each single filter.

The first phase (*init*) is initiated when the Spamato Base receives a message for which the spam probability has to be computed. The message is delivered by an email client add-on or intercepted if the Spamato Proxy is employed. After that, a `PreCheck` event is published to notify all interested plug-ins, especially the filters.

Generally, the purpose of the second phase (*pre-check*) is either to check if the message should be prevented from being filtered or to collect information useful to more than one filter. For instance, the Earl Grey filter's server component sends an email to a user in order to verify his email address. This specific challenge message is definitely no spam and, in this phase, the Earl Grey filter blocks the system from any further processing of this message.

Normal plug-ins other than filters can also subscribe to this event. For instance, a plug-in's task is to decide if

a message was revoked before. If so, it vetoes against further processing in order to prevent the message from being filtered out again. It is also possible rather to pre-process than to pre-check a message. For instance, a common URL identifying plug-in extracts all URLs in a message. Afterwards, it provides this information to all URL-based filters which in turn save the time and resources to do the same job redundantly.

If any filter vetoes against processing the message in the second phase, the process stops and the message is classified as ham. Otherwise, in the third phase (*check*), the message is scrutinized and each filter independently assigns the message a spam probability. In this phase, filters can also revert to pre-processed information collected in the pre-check phase. Since inspecting a message is a more complicated procedure than just to pre-check it, there are no time constraints on this phase. Still it is desirable to perform the spam check as fast as possible.

In the fourth phase (*decision*), the overall spam decision is calculated and sent to the Spamato Base which in turn forwards it to the user. Subsequently, the email client add-on or the Spamato Proxy moves the message to the special spam folder, if it is classified as spam, or leaves it untouched.

In the fifth and last phase (*post-check*), all registered plug-ins are provided with the final decision of the filtering process. The intention here is to enable filters to adapt to the overall outcome. For instance, in this phase, we automatically train our Bayesian filter by adjusting its good/bad token lists. While the plug-ins (filters) operate concurrently in the pre-check and check phase, the post-check phase is not time critical. Thus filters can sequentially be notified in order to save resources.

4 Filters

The aim of this chapter is to support our hypotheses that Spamato is a multi-faceted, extendable, and easy-to-use filter framework. The success rate of the filtering process exclusively depends on the quality of its filters. Therefore, the more filters of different techniques that exist, the better the overall filtering rate will be. The development of five different filters and their employment during several months of beta-testing shows the capabilities of Spamato. Developers can solely focus on the realization of their ideas instead of bothering about how to test and deploy their filters.

The aim of this chapter is not to claim some excellent results in the success rate of our filters. It is hard to corroborate such claims with less than 90,000 processed messages from a dozen users only. Nevertheless, we branded about 50,000 messages to be spam with a false positive rate less than 0.5 percent and about 7 percent false negatives.[2]

[2]Please note that we are still running a beta-test. We assume that the real rates are much better as test-cases currently tamper with the Statistics Engine.

In this chapter, we describe the underlying concepts, the advantages, and the drawbacks of our filters. The intention is to encourage developers to build even better ones or just to try out new ideas on top of the Spamato framework.

4.1 URL filtering

Spammers often advertise their products by referencing their web sites, which contain more specific information and, especially, order possibilities. The URL filtering technique is based on these references. Linked URLs or domains are extracted from an email in order to check if they have been blacklisted before. Blacklists, such as SURBL [18], can either be maintained by a single user or in a collaborative manner, consolidating the appraisements of possibly millions of participating users in an open database. URL-based filters are also a first class approach against phishing attacks since these emails definitely contain references to faked web sites. Naturally, URL-based filters do not work on messages which do not contain any URL.

It has been shown that spammers obfuscate their domain links to confuse users and filters and to elude identification [19], but this is only an algorithmic problem. Sophisticated algorithms are even able to cope with phishing attacks based on homographic similarities due to the support of Internationalized Domain Names.

Another problem emerges from the linking of multiple domains in an email (we call this a *multi-URL* message). Spam messages often contain URLs that are not related to the spammers' businesses. We investigated 13750 spam messages and discovered that about 5800 (42.2%) of them contained more than one URL and about 1000 (7.3%) even referenced ten or more distinct URLs. The reason for this enrichment of URLs is, for example, that spammers use images in their messages that are loaded from different righteous online shops or that they link to trustworthy sources to affirm their legitimacy. It is also a common practice to insert *fake* domains in a spam message for the sole purpose of misleading filters. For multi-domain messages, it is hard to determine the real spam domain(s) among all listed ones. This section continues describing three different approaches of URL-based filtering facing this problem.

4.1.1 Razor Filter (Whiplash)

Vipul's Razor [13] is a collaborative filter comprising two different techniques.[3] The *Whiplash* algorithm is URL-based and is discussed in the following; the hash-based *Ephemeral* algorithm is sketched in Section 4.2.3.

We want to emphasize, that we are neither the inventors nor the maintainers of the Razor network. But to the best of our knowledge, we have developed the first open-

[3]Some other techniques have been proposed. But they are either not open (but part of the commercial Cloudmark branch) or have been discarded due to high false-positive rates.

source Java implementation of Vipul's Razor filter that, as a part of Spamato, is much easier to employ than its console-based original written in Perl.

The Whiplash algorithm extracts all URLs from a message in order to check if the domains have been blacklisted before. The spam probability of each domain is evaluated by consulting the Razor network. If any of the domains is classified as spam, the whole message is classified as spam, too. We refer to this as the *"single-URL"* approach because it is based on the spam probability of each single domain.

The drawback of this approach is that when reporting spam messages to the Razor network, also ham domains probably contained in multi-URL messages are discredited. This means that, for example, a message which contains a single ham domain that was reported as part of a multi-URL message before, subsequently, is classified as spam, too.

4.1.2 Earl Grey Filter

The Earl Grey filter works collaboratively like the Razor filter. The spam probability of a message is derived from the global rating of linked domains. The Earl Grey filter is bundled with a set of components, such as a local and a global whitelist, to improve the filtering success. Additionally, a client-based reputation system prevents malicious users from manipulating the network.

In contrast to Razor's approach, the Earl Grey filter uses a *"multi-URL"* technique. This means that all URLs of a message are evaluated as a single entity. For this purpose, each unique domain of a multi-URL message is hashed (using MD5) and, afterwards, all hash values are summed up. The resulting *fingerprint* identifies the message and is looked up in the Earl Grey network to acquire its spam probability.

First, it is obvious that for messages which contain only a single URL both approaches are the same. There is no difference in evaluating a single domain or the hash of a single domain.

The Earl Grey filter is immune to the Whiplash problem described earlier. The fingerprint of a multi-URL message which contains one or more ham domains does not conflict with any other fingerprints derived from messages containing the same ham domains. But this approach bares another drawback. Just like messages interspersed with random text chunks paralyze a hash-based filter, the random insertion of constantly changing fake domains alters the fingerprint and makes it impossible for this filter to uniquely identify the message.

4.1.3 Domainator

The initial motivation for the Domainator was to alleviate the aforementioned drawbacks. By eliminating known ham and fake domains before verifying the remaining domains by the Razor or Earl Grey filter, their filter qualities should be improved. But instead of creating such a pre-checking tool, the Domainator now works as an independent filter.

The Domainator is a single-URL-based filter which queries Google's databases instead of maintaining its own. The queries sent to Google are twofold: On the one hand, we determine the number of web pages that reference the domain. This means, using the web interface of Google, we would enter something like *"ethz.ch"* (the *domain criterion*) in the search box and store the number of results shown in the header line. We are also interested in the number of web pages found for the domain and a key word associated with spam (the *domain+spam criterion*), such as *'ethz.ch spam'* or *'ethz.ch blacklist'*.

The idea behind this approach is that spam domains usually do not last very long and contain only a limited number of web pages so that Google is unable to index them. Additionally, most external citations are associated to spam related topics; several blacklists maintained by different users contribute to our search. Therefore, the ratio of both criteria will probably be near to one. On the other hand, well known ham domains are expected to result in many hits using the domain criterion and a low rate for the domain+spam query. Admittedly, we also have to deal with a few ham domains that have many hits in the domain+spam query due to their spam related nature, such as "spamassassin.org." Depending on a chosen threshold for the ratio of the spam+domain to the domain criterion, the false-positive rate can be adjusted in relation to the number of false-negatives.

To evaluate our assumptions, we have investigated 2276 domains found in messages that have been taken from the SpamAssassin *hard-ham selection* [20], actual domains from our Earl Grey database, and collected bookmarks from people using Spamato. We manually divided them into 781 *spam* domains, 312 *fake* domains, 1109 *OK* domains, and 74 *whitelist* domains. These categories have been chosen according to the following criteria: Spam domains are associated with the products advertised in a message. Fake domains have obviously been added to a message in order to confuse URL-based filters (invisible to the user). OK domains are trustworthy, "good" domains. And whitelist domains are domains of major companies like "microsoft.com" or "apple.com," which have been added to a global whitelist.

Figure 3 shows the result of the evaluation. Fake and OK domains are rather evenly spread over the whole spectrum and whitelisted domains result in numerous hits. As expected, most spam domains are significantly clustered in an area where other domains are rarely found (a low number of hits and most of them are spam related). In conclusion, the Google criteria provides a useful mean to distinguish between spam and ham domains.

4.2 Other Filtering Techniques

For completeness, in this section, we sketch three filters that do not follow any of the URL-based approaches described in Section 4.1.

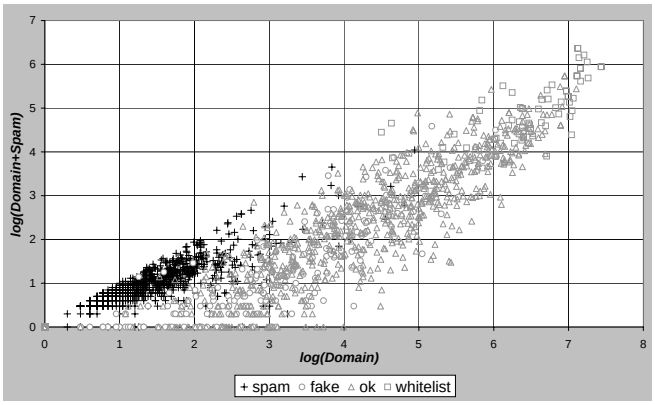

Figure 3: The evaluation of Domainator queries shows that spam domains can be distinguished from ham domains.

4.2.1 Bayesianato

The Bayesianato is a naïve Bayesian-based filter implemented according to Paul Graham's "A Plan for Spam" [21]. Since this technique is common to many filters and well-known, we will not describe it in more detail. It should be remarked, though, that the Bayesianato filter detects more spam messages than our other filters, but it has also the highest false-positive rate of all filters (which is still below 1 percent).

4.2.2 Ruleminator

The Ruleminator is as the name implies a rule-based filter. It allows to define logic rules, such as "if body contains 'sex' then spam" or "if 'X-SpamCheck' header begins with 'yes' then spam." Thus, it is similar to common filtering facilities of email clients but works on the Spamato layer.

An interesting capability is the explicit definition of ham messages. A built-in rule enables the filter in the *precheck* phase of the filtering process (see Section 3.4 for details) to veto against further processing of the message if the sender of the message has been seen before. Thus, this filter can establish an automatic whitelisting of known senders.

4.2.3 Razor Filter (Ephemeral)

The *Ephemeral* collaborative filter of the Razor system is hash-based. Small parts of the message body are hashed and the values (*digests*) are compared to entries in the Razor network.

The drawback of this approach is that the insertion of random text into a message deludes the filter as the calculated hash values are not identical. Still, the combination of the Ephemeral and the Whiplash (see Section 4.1.1 algorithms, leads to the excellent spam detection rate of the Razor system.

5 Conclusions

In this paper, we introduced Spamato as a multi-faceted, extendable, and easy-to-use filter framework. We showed how users and developers benefit from Spamato. Users can intuitively employ a single spam filter system integrated in their email clients. Developers can rely on a proven development environment to implement, analyze, and improve their filters without bothering about their deployment. Spamato is available for download at: `http://www.spamato.net`.

Acknowledgements

We thank Michelle Ackermann, Raphael Ackermann, Remo Meier, Simon Schlachter, Christian Wassmer, Andreas Wetzel, and all beta-testers for their contributions to the Spamato project.

References

[1] MessageLabs. Intelligence Annual Email Security Report 2004. http://www.messagelabs.com/binaries /LAB480_endofyear_v2.pdf.

[2] The Spamhaus Project. Increasing Spam Threat from Proxy Hijackers. www.spamhaus.org/news.lasso?article=156.

[3] Nicola Lugaresi. European Union vs. Spam: A Legal Response. In *Proceedings of the First Conference on E-mail and Anti-Spam*, 2004.

[4] Can-Spam Library. www.canspamlibrary.com.

[5] C. Dwork, A. Goldberg, and M. Naor. On memory-bound functions for fighting spam. In *Proceedings of Crypto 2003*, 2003.

[6] Microsoft. The Penny Black Project. http://research.microsoft.com/research/sv/PennyBlack.

[7] DomainKeys. http://antispam.yahoo.com/domainkeys.

[8] Sender ID Framework. www.microsoft.com/senderid.

[9] The Spamhaus Project. Jeremy Jaynes Gets 9 Years for Spamming. www.spamhaus.org/news.lasso?article=155.

[10] Procmail. www.procmail.org.

[11] SpamAssassin. http://spamassassin.apache.org.

[12] Theo Van Dinter. New and Upcoming Features in SpamAssassin v3. In *Talk at ApacheCon 2004*, 2004.

[13] Vipul's Razor. http://razor.sourceforge.net.

[14] Richard Segal, Jason Crawford, Jeff Kephart, and Barry Leiba. SpamGuru: An Enterprise Anti-Spam Filtering System. In *Proceedings of the First Conference on E-mail and Anti-Spam*, 2004.

[15] Cloudmark SafetyBar. www.cloudmark.com.

[16] SpamPal. www.spampal.org.

[17] XStream. http://xstream.codehaus.org/.

[18] SURBL - Spam URI Realtime Blocklists. www.surbl.org.

[19] Ken Schneider. Fighting Spam in Real Time. In *Proceedings of the 2003 Spam Conference*, 2003.

[20] SpamAssassin, Public Corpus. http://spamassassin.apache.org/publiccorpus.

[21] Paul Graham. A Plan for Spam. www.paulgraham.com/spam.html.