



Deliverable 5.2

Feature Extraction Report

Second Project Phase

K.U. Leuven

Version 02

November 05, 2007



IST 2006 027600

AntiPhish

Anticipatory Learning for Reliable Phishing Prevention

Specific Targeted Research or Innovation Project

2.4.3 Towards a global dependability and security framework

D 5.2 Feature Extraction Report

Due date of deliverable: M21 (30 September 2007)

Actual submission date: 9 November 2007

Start date of project: 01. Jan. 2006

Duration: 36 months

Lead Contractor for this Deliverable: K.U. Leuven

Revision: 02

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Revision history

Deliverable administration and summary		
Project acronym: AntiPhish		ID: IST-2006-027600
Document identifier:	AntiPhish-del-D52-FeatureExtractionReport-f-v02	
Leading partner: K U Leuven		
Report version: v02		
Report preparation date: 5 November 2007		
Classification: Public		
Nature: Report		
Author(s) and contributors: Jan De Beer, Marie-Francine Moens, in collaboration with all partners		
Status:		Plan
		Draft
		Working
	X	Final
		Submitted
		Approved

The AntiPhish © Consortium has addressed all comments received, making changes as necessary. Changes to this document are detailed in the change log table below.

Date	Edited by	Status	Changes made
-	DoW	Plan	report template
10.08.07	Jan De Beer	Working	First version
05.11.07	Gerhard Paaß	Final	Final edits

Notice that other documents may supersede this document. A list of latest public AntiPhish deliverables can be found at the AntiPhish webpage at www.AntiPhishResearch.org/publications.

Copyright

This report is © AntiPhish Consortium 2006. Its duplication is allowed only in the integral form for anyone's personal use for the purposes of research or education.

Citation

Jan De Beer, Marie-Francine Moens (2007). Deliverable D5.2 Feature Extraction Report. AntiPhish Consortium, c/o K.U.Leuven, www.antiphishresearch.org.

Acknowledgements

The work presented in this document has been conducted in the context of the EU Framework Programme project IST 2006 027600 AntiPhish. AntiPhish is a 36-month project that started on January 1st, 2006 and is funded by the European Commission as well as by the industrial partners. Their support is appreciated.

The partners in the project are Fraunhofer Institute for Intelligent Analysis and Information Systems (FHG), Symantec LIRIC Limited (LIRIC), Symantec Ltd. (Symantec Ireland), TISCALI Services S.r.l. (Tiscali) and K. U. Leuven / ICRI-LIIR (K.U. Leuven). The content of this document is the result of extensive discussions within the AntiPhish© Consortium as a whole.

More information

Public AntiPhish reports and other information pertaining to the project are available through AntiPhish public web site under www.antiphishresearch.org.

Table of contents

Executive summary	6
1 Introduction	7
1.1 Project Overview	7
1.1.1 Project Consortium	7
1.1.2 Work Packages	7
1.2 Feature Extraction in a Nutshell	8
1.2.1 State-of-the-Art.....	9
2 Activities.....	10
2.1 Software Platform	10
2.1.1 Updates	10
2.1.2 Additions.....	11
2.2 Feature Extraction.....	14
2.2.1 Updates	14
2.2.2 Additions.....	16
3 Conclusions and Future Work	22
4 References	23
5 Abbreviations/Glossary of Terms	24
6 Appendix A: Memory Requirements	26

Executive summary

During months 10–21 of the AntiPhish project, the activities in WP5 - the work package on message preprocessing and feature extraction - have concentrated on completing tasks T5.1 and T5.4 of Annex I to the Contract. These tasks address message salting, respectively structure and layout features. All such features have been successfully implemented and documented, and are available from the central code repository. Second, important improvements have been made in the underlying software platform, especially with regard to feature value caching, distributed operation, configuration and setup. Third, minor updates to previous feature implementations have been applied as the result of a continuous quality assurance process between the K.U.Leuven work package coordinator and the other project partners, in particular the work package participant FHG.

Noteworthy is the pioneering work done in the area of message salting. We were able to invent and implement new technology for the detection and resolution of hidden text salting tricks. The technology is being applied for patenting.

Lastly, the work package participants have kept up with literature, integrating features that represent the state of the art in the field of email filtering. Their integration allows for a comparison with contemporary related work, and benefits from its contributions.

1 Introduction

1.1 Project Overview

Over the years email users have grown accustomed to spam; the electronic equivalent of bulk mail. Although regarded as inappropriate and intrusive, spam is by most considered not more than a nuisance they have learned to live with. However, there is a new form of spam waiting in the wings which may well dramatically change this opinion.

Phishing messages come from criminals seeking to deceive recipients into surrendering sensitive, personal data that is then used to perpetrate identity theft. Using techniques of social engineering and spoofing, phishers go to the extreme to imitate authoritative, trustworthy parties. In a constant arms struggle with spam filters, they remain one step ahead, exploiting the technicalities of the communication medium used to unmatched levels of ingenuity.

In the 3-year STREP EU-funded AntiPhish project (EU Sixth Framework Programme), a consortium of world-leading research and industrial partners have taken up the challenge of developing anticipatory spam and phishing filters of the next generation. AntiPhish will address the aforementioned deficiencies by developing highly performant, highly accurate, trainable and adaptive filters that are not only able to identify variations of previous phishing (spam) messages, but that are capable to anticipate new forms of phishing (spam) attacks. Currently, such filters do not exist, and out-of-the-box machine learning algorithms are not able to handle this problem in a satisfactory way.

Innovations in content analysis and information extraction technologies, in advanced machine learning algorithms, and in their tight coupling will produce the filters necessary to establish more secure and dependable communication networks.

1.1.1 Project Consortium

The partners in the project are Fraunhofer Institute for Intelligent Analysis and Information Systems (FHG), Symantec LIRIC Limited (LIRIC), Symantec Ltd. (Symantec Ireland), TISCALI Services S.r.l. (Tiscali) and K.U.Leuven Dept. of Computer Science (K.U.Leuven).

1.1.2 Work Packages

The work is divided into eight work packages. WP1 provides the data which will be used to train our filter. In WP2 the requirements of the system will be analyzed and specified, while WP3 will deal with the actual integration and validation of the system. WP4 aims at adapting and testing filters in a wireless environment. WPs 5 and 6 deal with the scientific aspects of AntiPhish. While WP5 concentrates on the preprocessing and feature extraction, WP6 develops the actual learning algorithms. WPs 5 and 6 are closely related, as the learning algorithm and the features have to be adjusted and tuned with respect to each other. WP7 deals with the dissemination and exploitation, while the overall management is contained in WP8. The structure and dependencies between all work packages is depicted in Illustration 1.

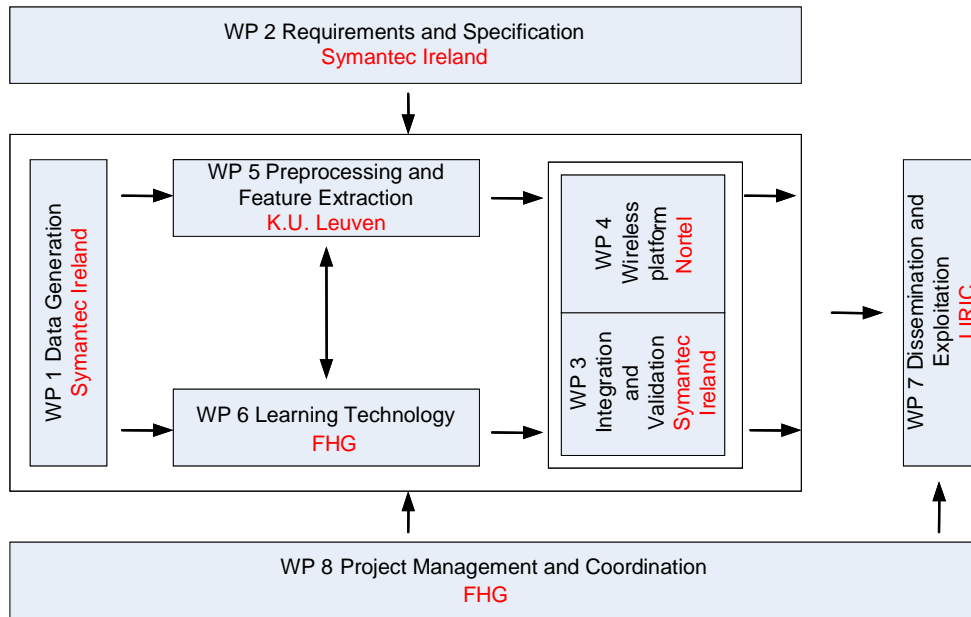


Illustration 1: Work package Pert diagram.

1.2 Feature Extraction in a Nutshell

The core objective of the work package on message preprocessing and feature extraction (WP5) is to design representations of messages that are better suited to phishing filtering, together with the software programs to automatically generate these representations from the raw message data. These representations will then be operated on by the advanced learning algorithms developed in WP6.

We extend from the traditional bag-of-words approach to a more comprehensive set of basic and advanced (technically challenging) features. As outlined in Annex I to the Contract, we envision several feature classes: salting, syntactic, semantic, structure and layout, link-related, topical, graphical, and multi-message. Each class contains several features which will be realised using state-of-the-art or innovative extraction technology.

A first cross-cutting concern for the design and implementation of our feature extraction system is to have reliable and fast implementations that can stand the challenges of a filtering system working online in real time at the service provider. To achieve this, the extraction algorithms will be continuously evaluated for quality, robustness, speed and scalability. Improvements and revisions of the software will be made accordingly.

A second cross-cutting concern is to have a flexible and easily extensible software architecture. The architecture should allow support for various (new) message types, message feeds or repositories (i.e., static vs. streaming, offline vs. online), feature sets, pluggable, often locale-dependent knowledge or processing resources (e.g., natural language processing modules and lexicons), and filtering tasks. While we are committed

to make our software architecture open-ended, at first full support is being developed for the filtering of phishing Internet Messages (email), with content analysis for the English language (including content embedded in images).¹

A third and last cross-cutting concern, which lies at the heart of the project's aims, is the anticipation of new emerging spam or phishing tricks well ahead in time. This asks for an open-ended feature set and anticipatory extraction/learning technology, which would mean raising the state-of-the-art in these research areas.

1.2.1 State-of-the-Art

Current systems that filter phishing emails mainly rely on structured features extracted from the email header or from the body of the texts. The features extracted from the header are well-known (e.g., the header fields 'title', 'from', 'to'). The structured features from the body are often suspicious domain names of links (e.g., [Kirda et al. 2006]).

When unstructured contents of the emails are considered, state-of-the-art tools rely on character sequences (n-grams), the words of the emails, or word tuples composed of two or more words possibly separated by wildcards (e.g., [Assis et al. 2005]). But, the filters ignore features based on the layout of the emails, syntactic structure, and semantic or topical classifications of the content. They completely neglect hidden salting tricks which accomplish that the filter considers different content from what the human user of the email perceives. In addition, they ignore the dynamic adaptation of the feature extraction methods to new scams.

In their current research, the AntiPhish Partners have already moved beyond this state-of-the-art by implementing the extraction of syntactic, structural, layout, and salting features.

¹ *Deployment in a wired or wireless networking environment at a later stage of the project may require developing support for additional message types and/or languages.*

2 Activities

In this section we give an overview of the main activities and achievements in WP5 during the second year of the AntiPhish project.

2.1 Software Platform

Regarding the feature extraction (FX) software platform, we highlight the most important updates and additions.

2.1.1 Updates

As for the email message parsing, our parser has been made more robust. As input, it accepts email data files (.eml files) that are formatted as either UNIX or DOS. The difference between these two formats arises from a different convention in the control sequences that mark line endings. This update eliminates the need for explicit file format conversion when exchanging files between project partners and applications. Also, the parser is more lenient to malformed messages. Through minor updates, we were able to reduce the parsing error rates reported in D5.1 [De Beer et al. 2006] from 0.84% to 0.81% for ham, from 1.97% to 1.65% for phishing, and a significant reduction from 45% to 0.95% for regular spam.

As for the email message representation, we identify several timestamps. Since the 'send:' timestamp included in the email header is found to be unreliable (inaccurate or forged) especially in spam emails, we also provide the timestamp of the firstmost (most recent) 'received:' header field. Since the 'received:' header fields are added by intermediate mail agents on the Internet (outside of the sender's control), their values are more reliable. Timestamps will become important for the multi-message feature extraction task (T5.7) as well as for evaluations involving simulated message streams.

As for the caching of computed feature values, we have redesigned the caching framework for increased flexibility and control (cf. Illustration 2). In particular, the caching storage medium is separated from the caching strategy (i.e., the policy specifying what is to be cached and what needs to be purged). We have provided two medium implementations: memory (fast but small) and disk (slower but larger and more easily extensible). As a caching strategy, we provide a least-recently-used (LRU) implementation with customizable cache size. Lastly, fine-grained control over caching is realised by enabling/disabling the volatile (memory-based) and persistent (disk-based) caching of feature values on a per-feature-type basis, as can be explicitly specified in a configuration file.

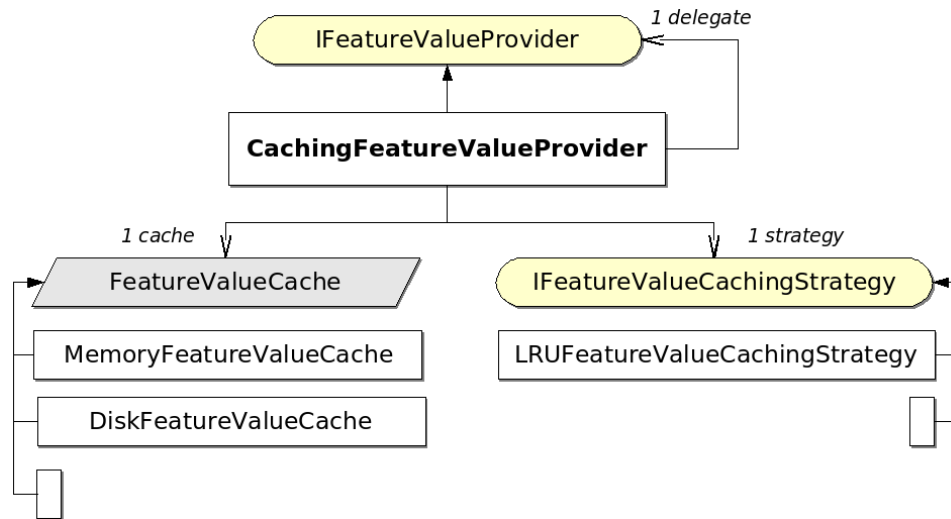


Illustration 2: Class diagram of the renewed feature value caching framework. The diagram shows the separation between the cache storage medium (left) and the caching strategy (right), united in a `CachingFeatureValueProvider`.

2.1.2 Additions

In line with the caching redesign, we provide support for a more efficient caching when performing feature extraction in a distributed computing environment. In particular, we envision an environment in which multiple, interconnected computing nodes require or generate sets of feature values that may partly overlap. To prevent or limit the redundancy in computation that may arise from overlapping needs, we provide a global cache, which is managed and located at a dedicated node, which is referred to as the 'server' node. The global cache is shared by the other 'client' nodes. Sharing comprises the publishing of computed feature values at and by the individual client nodes to the server node for future retrieval (by possibly other nodes), and a hierarchical cache probing mechanism (cf. Illustration 3), that considers the server's global cache after the requesting node's own private cache. The nodes' private caches serve as proxies (fast, local caches) to the global cache, and can be tailored to the nodes' individual needs (e.g., by means of a custom caching strategy) and the availability of memory and storage space.

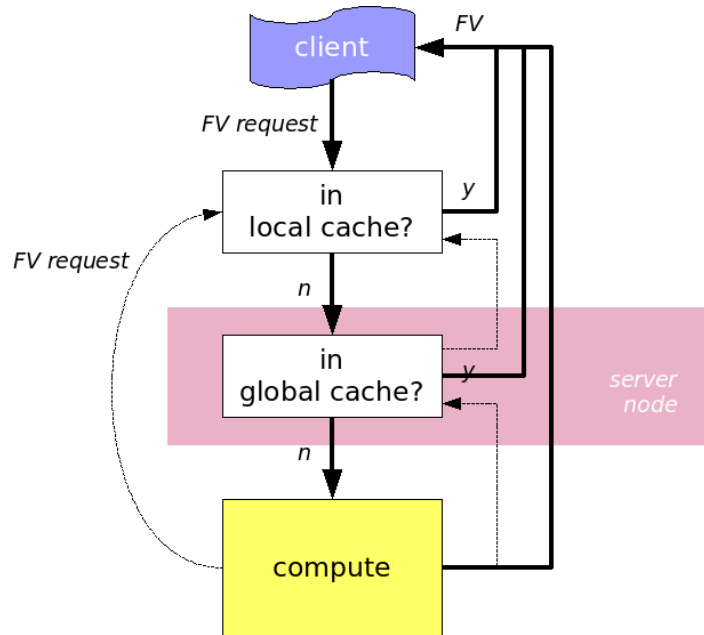


Illustration 3: The hierarchical feature value cache probing mechanism in a distributed (client-server) environment. The diagram starts with the issuing of a feature value (FV) request by a client (top). In subsequent phases, the local cache, global cache, and computing FV provider are consulted in an early-out fashion. A retrieved or computed FV is cached on lower levels (thin straight loopback arrows) before reaching the client (thick loopback arrows). The computation of a FV may itself spawn other FV requests (thin curved loopback arrow).

As more feature types and their computing providers are (being) added to the platform, more diverse caching schemes become available, and distributed operation is supported next to single-node operation, the need arises to define proper default installations to bootstrap the feature extraction process. These installations, termed *FX environments*, facilitate the setup and usage of our software platform by hiding all the complexities involved (e.g., creating providers, linking or aggregating providers, introducing caching and distributed operation). By providing a uniform, high-level API of feature value provision, novices to our platform immediately benefit from best practices, and can quickly build up familiarity with our software using the predefined FX environments. Whenever required, ultimate control is made possible through customizable configuration files, customizable providers, and the arbitrary assembly and usage of providers by client software.

Currently, we provide two predefined FX environments.

- The 'default' environment is specifically designed for single-node feature extraction. It makes use of a local, private cache.
- The 'distributed' environment implements the client-server, distributed computing environment (cf. supra). Both client nodes and server node receive a

private cache. In addition, the server node manages a shared, global cache. The distributed caching implementation makes use of Java RMI (Remote Method Invocation) technology.

Lastly, we have added dynamic FX configuration support to our software platform. We hereby refer to the set of parameters that influence and configure the generation (extraction) of feature values. For example, the generation of the 'n-grams' feature values (O.ngr, cf. Illustration 4) takes as a parameter the value of 'n'. Previously, all parameter values were assumed static (i.e., constant during the entire runtime of the platform). For example, the assumption implied that the platform could not generate both bigrams (n=2) and trigrams (n=3) of a message's contents. This restriction has been removed in the current version of our platform. Most of the FX parameters can now be made dynamic; their values can be altered at runtime. Details are provided in the platform's Javadoc documentation.

2.2 Feature Extraction

This section presents our progress on the feature extraction tasks. Both updates to previously available features, as well as new feature types are presented and discussed. Illustration 4 gives an overview of all implemented features so far and their dependencies. An evaluation of their memory requirements can be found in Appendix A.

2.2.1 Updates

In close collaboration with FHG, the following updates were proposed and successfully implemented.

- Previously, the 'message text' (PRE.txt) primitive feature value represented the naïvely extracted text contents from a message, i.e., the textual data that remains after stripping all structural and markup information from the message data (cf. D5.1 [De Beer et al. 2006]). As more sources of textual content become available, we allow the specification of one or more alternative (upon failure) or concatenated textual data sources for the 'message text' (PRE.txt) feature value. Currently, the following sources are defined.
 - The message subject line, which is taken from the corresponding header field (assumed to be plain text).
 - The naïvely extracted message body text (cf. supra).
 - The perceived message text (cf. infra; our salting solution T.sal)
 - The text extracted by the application of OCR (Optical Character Recognition) to the images contained in the message, if any.

Hence, the textual content analysis performed by features that are directly or indirectly dependent on PRE.txt (e.g., the morphologic-syntactic features, see Illustration 4) can be applied to whatever content source desired.

- The data representations of the 'characters' (O.chr) and the 'n-grams' (O.ngr) features is changed from a character tree representation to a flat lookup table. The new representation is more economic in terms of memory, and is faster to create and inspect.
- By the implementation and usage of a robust and fast scanner that sequentially runs over HTML contents, the message 'body parts structure' (T.str) feature value has a better recall of links, in particular URL hyperlinks to referenced Web sites or external images.

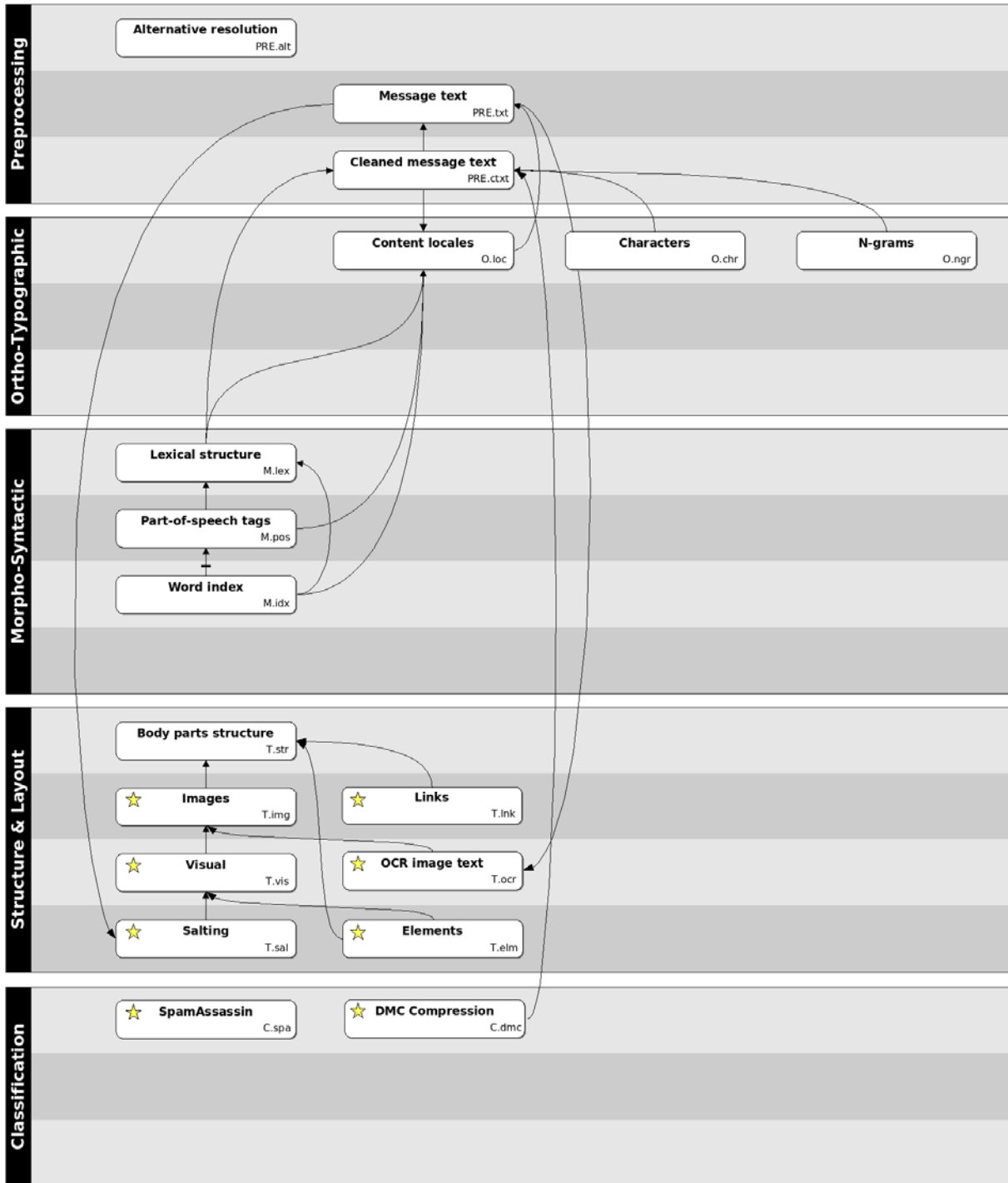


Illustration 4: Diagram showing all implemented features. Every box represents a single feature and displays its name and identifier. New features are marked with a star. Arrows between boxes denote dependencies between features. For example, cleaning the message text (PRE.ctxt) requires the original text (PRE.txt) and its locale (O.loc).

2.2.2 Additions

We have completed the tasks T5.1 and T5.4, described in Annex I to the Contract, by the implementation of features that relate to message salting, respectively message structure and layout. For the tasks T5.3 (Semantic Feature Extraction) and T5.6 (Graphical Information) we developed new features, which are intimately related to machine learning methods and are described in detail in [Paass et al. 2007]. We continue by describing the individual features.

- T.img represents the images encapsulated or referenced by a message (i.e., both internal and external images). Internal images are decoded from the raw message data and instantiated in memory as instances of the Java superclass `java.awt.Image`. As a configurable option, external images may be either fetched over the network and instantiated in memory, or discarded. Discarding external images is the recommended setting since the transfer of image data may introduce substantial latencies (in the order of seconds), is not guaranteed to succeed (e.g., network ruptures, moved or unresponsive servers), raise security issues (e.g., insecure network connections to the hostile Internet are made), and may be spoofed or unreliable (e.g., servers may produce different images on subsequent requests, both for legitimate purposes as for deception).

A full image of the entire message view (e.g., a rendering of an email as in a graphical email client) is provided by the 'visual' (T.vis) feature value, described next.

- T.vis represents the visual representation of a message, which is an HTML page transcribed from the message contents for the purpose of visualisation. We choose for HTML as the visual representation medium since it is a rich medium (supporting formatted text, tables, forms, images, links,...) that is commonly used in email, is easy to generate and parse, and has sufficient support in the standard Java distributions. In Java, HTML pages can be parsed into a graphical component of type `java.awt.Component`, which can be rendered to screen or some other output device. By rendering to a newly created image of corresponding size, we create a *view* of the message; a full image (screen capture) of the visual representation of a message.

As a configurable option, the message content media that are included in the transcription of the HTML page can be specified (by default, plain text and HTML text parts are included). Also, the HTML page may reference the original, transcribed message images or reference a fixed substitute image of the corresponding sizes. This option allows the use of a (blank) filler image, eliminating the need for transcribing image data (a costly disk operation), while preserving the page layout.

- T.sal represents the results of the hidden text salting analysis applied to a message. It comprises both the perceived text contents of the message (free of any hidden text salting) as it would be observed and read from the message view by humans, and counts on the number of detected salting tricks for the trick types listed below.
 - The use of invisible or hardly visible *font colour*.
 - The use of illegible *font size*.

- Text made invisible by placing it outside the visible rendering region (exploiting a rendering technique known as *clipping*).
- Text made invisible by *concealing* it with other text or opaque, overlapping shapes.
- Text that is read in a different order than it is rendered or, equivalently, contained in the message source data. Spammers are known to exploit the naïve assumption of conventional text extraction methods for email filtering. The assumption made is that accumulating text contents from a message's source data in a simple forward pass through that data yields equivalent textual contents than those perceived by the end user from the on-screen displayed message. Using hidden salting tricks such as the 'slice-and-dice trick' [Cumming 2003], the *text order* assumption can be broken, generating noise for filtering purposes, yet at the same time conveying their fraudulent or unsolicited message to the end user (cf. Illustration 5).

The salting analysis is based on a portion of the visual representation of the message (*T.vis*). In particular, for performance reasons, the analysis is restricted to the message *page*; the topmost part of the message view. In the case of email, the page corresponds to the initial visual part of the email message when opened in a graphical email client. The preferred and maximum page dimensions can be configured. Furthermore, a timeout can be specified in order to abort the analysis in the rare cases (0.02% in a test email corpus) when the analysis freezes. The freezing is due to some unidentified problem in Java's HTML libraries. When becoming problematic for future data sets, we can freely switch to an alternative library.

The concepts and methods that make up the salting analysis (solution) are fully described and evaluated in a scientific journal article, submitted for publication [De Beer et al. 2007]. Considered innovative technology, we have applied for a patent. Both submissions are already available for members of the Consortium only, including the Commission Services. A presentation of concepts and a first prototype implementation of the technology were demonstrated at the first review meeting (November 2006).

We have evaluated the intrinsic quality, utility, and performance of our hidden text salting solution on a representative (private) corpus of email messages, described in D5.1 [De Beer et al. 2006]. The results and a discussion are available from [De Beer et al. 2007]. To illustrate, Table 1 contains the summary results on the prevalence of the detected hidden text salting tricks (cf. *supra*) for the entire email corpus. The corpus is a superset of the TREC 2001–2002 public email corpus, which has been commonly used in spam filtering studies. Its results are distilled into Table 2. Disregarding the 'text order' trick (which is not yet evaluated and deemed less accurate), the tables show that at least 3–4% of all spam and phishing messages contain some type of hidden text salting (2–3% is substantially salted), while substantial hidden text salting is hardly found in legitimate messages.

The integration of our salting solution in the advanced learning technology of WP6, allowing for extrinsic evaluations (i.e., measuring the effectiveness of the solution for filtering purposes), will be reported on in D6.2; the second year prototype report.

- T.Ink provides information on the links in a message. Inspired by [Fette et al. 2007], the feature currently comprises counts on the number of internal links,

external links, external links having an IP-based host specification (e.g., '66.211.168.65' instead of 'www.paypal.com'), and external links having a host specification that is different from the host mentioned in the link text (e.g., '[www.paypal.com](#)' in the link text but '[www.spoofedsite.net](#)' in the source data). In addition, for the external links (URLs), we provide a service interface for the implementation of URL black- and whitelists.

- T.elm provides information on certain structural or layout elements in a message. Inspired by [Fette et al. 2007], the feature currently comprises the presence or absence of scripting language (such as JavaScript) or input forms in the message source data.
- T.ocr represents the results of an OCR engine applied to the images in a message, as inspired by [Fumera et al. 2006]. Currently, the feature value comprises the plain text extracted by means of OCR from the images. The feature T.ocr is complementary to the salting feature T.sal, in that the former provides the perceived text of image parts, whereas the latter provides the perceived text of textual content parts. In a later stage, a unified approach may be worked out, as is foreseen in the salting solution. For instance, to anticipate tricks where the perceived text runs over both, the image and the textual parts, each part representing a fragment of the perceived text.
- C.spa represents the classification outcome of the popular open-source SpamAssassin email filtering program [SpamAssassin]. This feature allows for a comparison with [Fette et al. 2007]. Furthermore, it allows SpamAssassin to be used as a baseline classifier or as one classifier in a meta-learning setting (bagging, stacking, boosting), and may improve classification accuracy especially in the separation of ham and spam email.
- C.dmc represents the classification outcome of the Dynamic Markov Chain compression model; a robust technique for email (spam) classification that was recently proposed in [Bratko et al. 2006]. In short, the authors apply adaptive statistical data compression models to raw email data (as binary or character sequences), making sensitive preprocessing steps unnecessary. Adaptively building up a separate model for classified ham and spam email, the classification outcome of a new target email is determined by the model that yields the best compression rate on the target email. As measures of compression, the authors use the cross-entropy and the description length.

The benefits of this feature are analogous to C.spa.

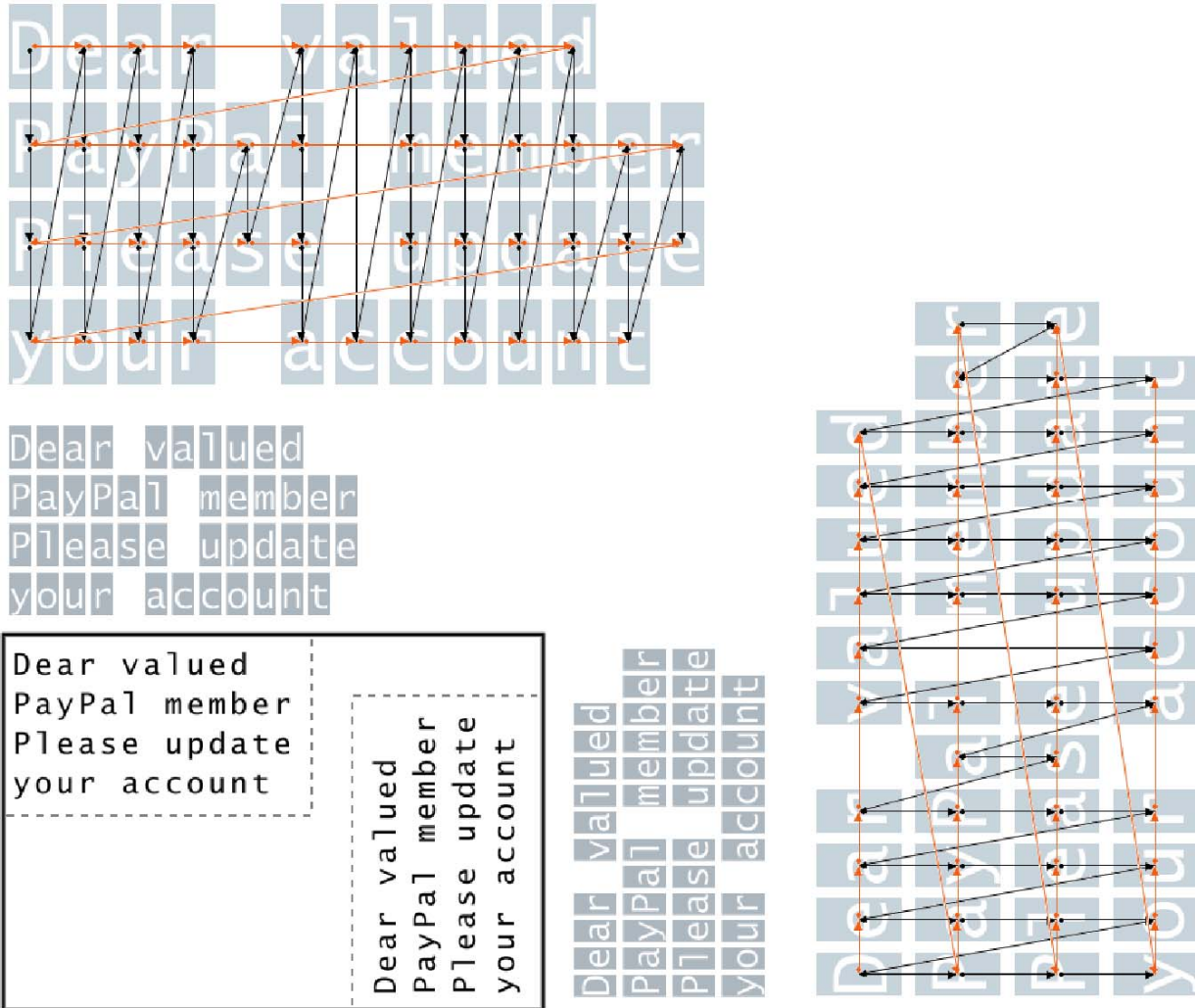


Illustration 5: The concept of text reading and rendering order, and how the latter order is dubious. The bordered frame outlines a page containing two text blocks (shown dashed), whose perceived contents arise from rendering a virtual grid of text characters (shown asides). The grid traversal order during rendering can be manipulated, resulting in a text rendering order (dark arrows) that differs from the text reading order (light arrows). Furthermore, characters' rotation angle within their grid cell can be set at will (not illustrated). The salting solution which we have invented is robust against these tricks.

<i>r</i>	Message category	Clipping	Concealment	Font colour	Font size	Text order	Any trick	Any trick but order
> 0%	Ham	0.0	3.7	9.5	2.3	55.1	60.9	14.6
	Spam	0.2	3.4	30.0	5.7	119.2	143.4	36.4
	Phishing	0.0	0.0	34.8	3.6	225.8	245.4	37.5
	All	0.1	3.6	17.7	3.7	81.6	94.7	23.3
≥ 1%	Ham	0.0	0.3	6.7	1.0	24.1	30.7	7.6
	Spam	0.2	2.3	27.9	5.3	84.9	110.9	33.3
	Phishing	0.0	0.0	34.4	2.7	105.8	136.1	37.0
	All	0.1	1.0	15.2	2.7	48.5	62.9	17.9
≥ 10%	Ham	0.0	0.2	1.5	0.1	4.9	6.9	1.8
	Spam	0.0	0.2	21.4	4.4	28.2	53.7	26.2
	Phishing	0.0	0.0	29.0	1.8	8.0	38.8	30.8
	All	0.0	0.2	9.5	1.8	14.0	25.4	11.5

Table 1: The prevalence of detected hidden text salting tricks in all 252515 English messages of the private email corpus described in D5.1, by message category and trick type. The numbers represent messages per thousand. Aggregates are provided over message categories ('All' messages) and trick types ('Any' tricks, with and without 'text order'). The table differentiates three salting degrees '*r*', defined as the percentage of characters in the analysed message page that are affected by the detected tricks. The corpus' spam proportion is 39%, the phishing proportion is 0.9%. The proportion of HTML messages (as common source of salting) is 19% for ham, 53% for spam, 93% for phishing, and 32% in total.

<i>r</i>	Message category	Clipping	Concealment	Font colour	Font size	Text order	Any trick	Any trick but order
> 0%	Ham	0.0	0.3	1.9	0.0	21.1	21.3	2.2
	Spam	0.0	1.3	18.1	9.1	116.9	134.9	26.9
	All	0.0	0.9	11.2	5.2	76.3	86.7	16.4
≥ 1%	Ham	0.0	0.2	0.3	0.0	7.6	8.2	0.4
	Spam	0.0	1.0	15.9	8.9	90.9	112.0	24.6
	All	0.0	0.7	9.3	5.2	55.6	68.0	14.4
≥ 10%	Ham	0.0	0.0	0.0	0.0	0.8	0.8	0.0
	Spam	0.0	0.3	11.6	8.0	32.5	51.7	19.9
	All	0.0	0.1	6.7	4.6	19.1	30.2	11.4

Table 2: The prevalence of detected hidden text salting tricks in the 2001–2002 TREC public email corpus containing 92189 messages. Cf. Table 1. The corpus' spam proportion is 57.3%. The proportion of HTML messages is 8% for ham, 53% for spam, and 34% in total.

3 Conclusions and Future Work

At the end of month 21 of the AntiPhish project, the WP5 participants K.U.Leuven and FHG have succeeded in completing tasks T5.1 to T5.4, delivering high-quality implementations and documentation for the features covered by these tasks, as well as for the enabling software platform that generates these features. Keeping up with literature, features were integrated that reflect the latest advances in the field of email filtering.

Due emphasis and time was given to the salting feature. This feature turned out to be the most challenging to capture due to its diversity and sophistication, its novelty, and highly competitive nature. We are under the impression that research in this area is scarce and not fully developed, whereas the need for a general solution highly desired and pertinent. By our invention of a general and effective solution to hidden text salting, we believe that we have considerably contributed to this field.

As for the remaining tasks and project year, a working document has been drafted by the WP5 participants, presenting in concrete terms an overview and a planning of the remaining work and objectives. The document is available for members of the consortium (including the Commission Services) from the private BSCW document management server. The participants remain confident in realising all of AntiPhish's goals.

4 References

[Assis et al. 2005] Fidelis Assis, William Yerazunis, Christian Siefkes, and Shalendra Chhabra. *CRM114 versus Mr. X: CRM114 Notes for the TREC 2005 Spam Track*. In TREC 2005 Workbook, 2005.

[Bratko et al. 2006] A. Bratko, G. Cormack, B. Filipic, T. Lynam, B. Zupan, Spam filtering using statistical data compression models, *Journal of Machine Learning Research* 7 (2006) 2673–2698.

[Cumming 2003] J. Graham-Cumming, The spammer's compendium, in: Proceedings of the 2003 Spam Conference, 2003, kept updated at <http://www.jgc.org/tsc.html>.

[De Beer et al. 2006] Jan De Beer, Marie Francine Moens (2006). Deliverable D5.1: Feature Extraction Report. Working version. AntiPhish Consortium, c/o K.U.Leuven, www.antiphishresearch.org

[De Beer et al. 2007] J. De Beer, M.-F. Moens, Detecting and Resolving Text Salting through Rendering Analysis and Cognitive Processing, in: *ACM Transactions on Dependable and Secure Computing* (submitted), 2007.

[Fette et al. 2007] I. Fette, N. Sadeh, A. Tomasic, Learning to detect phishing emails, in: Proceedings of the 16th International World Wide Web Conference, 2007.

[Fumera et al. 2006] G. Fumera, I. Pillai, F. Roli, Spam filtering based on the analysis of text information embedded into images, *Journal of Machine Learning Research* 7 (2006) 2699–2720.

[Kirda et al. 2006] Ergin Kirda, and Christopher Kruegel. *Protecting Users against Phishing Attacks*. *The Computer Journal*, 2006, 49(5):554-561.

[Paass et al. 2007] Gerhard Paaß, Andre Bergholz, Jeong-Ho Chang, Thorsten Merten, Anja Pilz, Frank Reichartz, Siehyun Strobel, Kejun Xu (2007). First Phase Preliminary Results Report. Fraunhofer IAIS, AntiPhish Consortium, www.antiphishresearch.org.

[SpamAssassin] The Apache SpamAssassin project. <http://spamassassin.apache.org>

5 Abbreviations/Glossary of Terms

BSCW: Basic Support for Cooperative Work

Document management tool for collaborative work, in use by the AntiPhish Consortium.

FX: Feature Extraction

The derivation of distinguishing feature values (characteristics) from objects of interest. Refer to D5.1 [De Beer et al. 2006] for an introduction to feature extraction.

HTML: HyperText Markup Language ⇒

hypertext markup language (HTML)

The Internet's primary markup language. The language defines a set of formatting and structuring tags for marking up text.

IP: Internet Protocol ⇒

ham

The category of legitimate, wanted messages. Compare to *spam*.

hidden salting

Introducing noise in a message which is not visible in the message's visual representation. It is used for countering current message filtering techniques. Compare to *surface salting*.

internet protocol (IP)

A network layer protocol in the Internet protocol suite. As a lower layer protocol, IP provides the service of communicable unique global addressing amongst computers.

java

Refers to the Java programming language or Java technologies. More information on <http://java.sun.com>

OCR: Optical Character Recognition ⇒

optical character recognition (OCR)

Computer methods designed to translate images of typewritten text into machine-editable text.

phishing

Phishing messages come from criminals using counterfeit identities to deceive recipients into surrendering sensitive, personal data that is then used to perpetrate identity theft.

RMI: Remote Method Invocation ⇒

remote method invocation (RMI)

The Java technology allowing communication between remote (i.e., residing in different Java virtual machines) Java objects. Constitutes the Java equivalent of RPC (remote procedure calls). More information on RMI can be found at:

<http://java.sun.com/j2se/1.5.0/docs/guide/rmi/spec/rmiTOC.html>

software platform

The entire bundle of source code, binary code, and documentation that is used for the automated processing of messages in the AntiPhish project.

spam

The category of unsolicited, often intrusive messages. Compare to *ham* and *phishing*.

surface salting

Introducing noise in a message which is still visible in the message's visual representation. It is used for countering current message filtering techniques. Compare to *hidden salting*.

URL: Uniform Resource Locator ⇒

uniform resource locator (URL)

A pointer to a 'resource' on the World Wide Web. Colloquially referred to as a (hyper)link. More information on the types of URLs and their formats can be found at: <http://www.socs.uts.edu.au/MosaicDocs-old/url-primer.html> (an example URL)

WP: Work Package

6 Appendix A: Memory Requirements

We have evaluated the runtime memory requirements of our feature extraction (FX) software, run on a 32-bit computer architecture. In the experiment, we aim to measure the amount of memory that is needed for the extraction and the representation of the different features for a single message. The formula given below extrapolates to multiple messages. As it is the recommended setting, any dependent features (cf. Illustration 4) are cached for performance reasons, and thereby included in the measure of the target feature. As target features, we consider a selection of features at important intermediate or ending (aggregate) positions in the FX network of Illustration 4. As our message test corpus, we use a subset of the email corpus described in D5.1 [De Beer et al. 2006]. The test corpus is composed as follows: 3846 ham messages (45%, chosen randomly), 2492 spam messages (29%, chosen randomly), and all 2280 phishing messages (26%), totalling 8618 messages.

In the experiment, we distinguish three sources of memory consumption. First, the *base memory* is the memory that is consumed by the initial setup of the FX software platform. The amount of base memory m_b is a constant, measuring 2.8 megabytes for the default FX environment (cf. Section 2.1.2) that is used in the experiment. Second, M_m is a stochastic variable that denotes the amount of memory that is used for the internal (object-oriented) representation of a single message. In particular, it comprises all header and metadata information, the structural composition (i.e., MIME parts) and location information (e.g., pointers to file contents) of the message. The actual message data however, is excluded (e.g., resides in a disk file). Third, $M_f(x)$ is a stochastic variable that denotes the amount of memory that is used for the extraction and representation of feature x and all dependent features (cf. supra) from a single message.

Given m_b , M_m , and $M_f(x)$, the expected total memory requirement $E[M]$ for the extraction and representation of feature x for n messages (caching all results in memory), is then given by

$$E[M] = m_b + n \cdot (E[M_m] + E[M_f(x)])$$

The actual memory requirement M may be smaller due to resources shared between the messages (e.g., message representation common data structures), or larger due to increased overhead (e.g., expanded caching data structures).

In the figures that follow, we plot the cumulative distribution functions of M_m (Illustration 6) and $M_f(x)$ (Illustration 7 to 12) for the messages in our test corpus and the selected target features x . The memory measures are best interpreted as rough approximations, due to Java's dynamic memory management processes (e.g., automatic garbage collection and chunkwise memory allocation/release). For the salting feature ([T.sal](#)), there is an additional memory requirement of 128MB to perform the analysis.

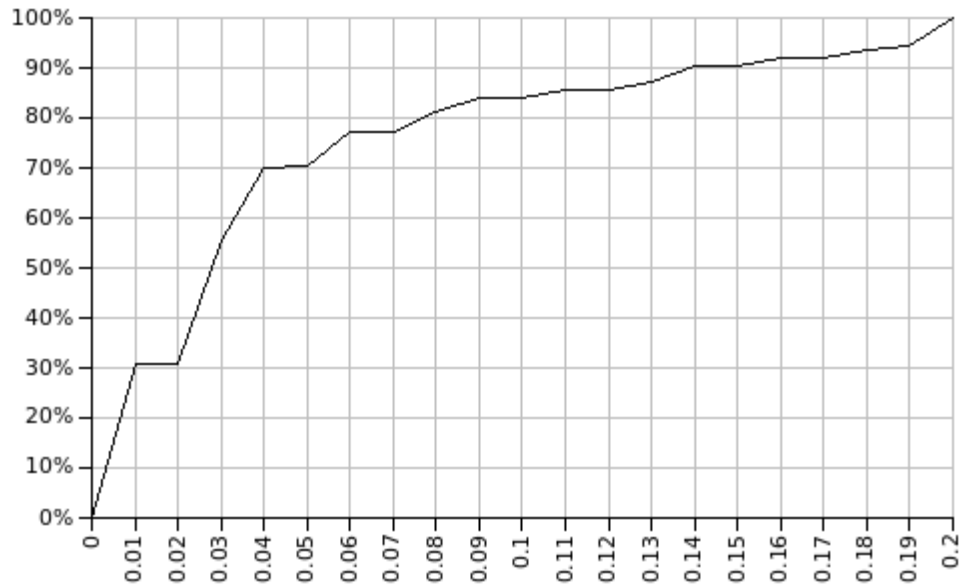


Illustration 6: The cumulative distribution function of M_m . The labels on the value axis indicate the lower bounds of 10KB-sized memory bins, in megabytes. The figure shows that for 80% of all messages, the internal message representation consumes less than 80KB of

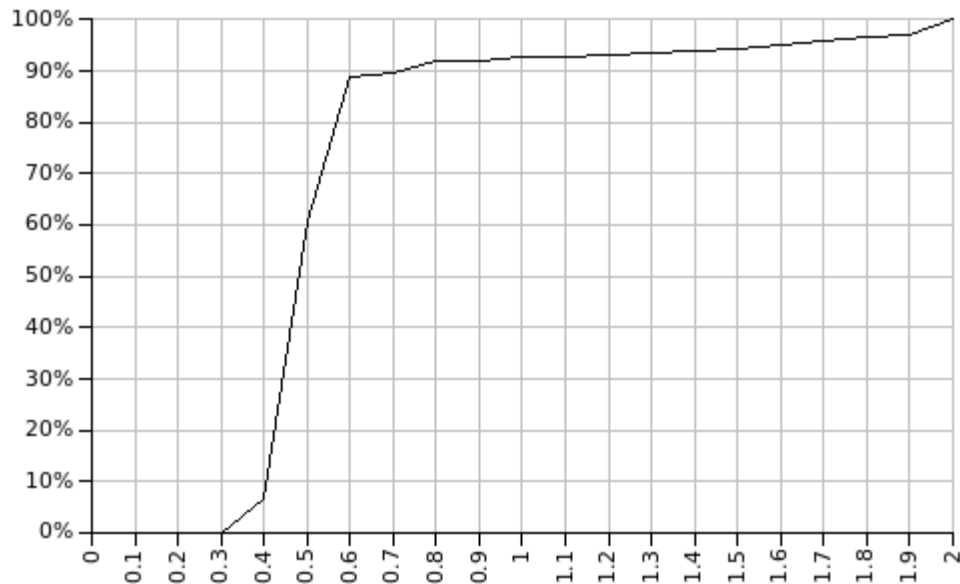


Illustration 7: The cumulative distribution function of $M_r(\text{PRE.txt})$; the 'cleaned message text' feature. The labels on the value axis indicate the lower bounds of 100KB-sized memory bins, in megabytes.

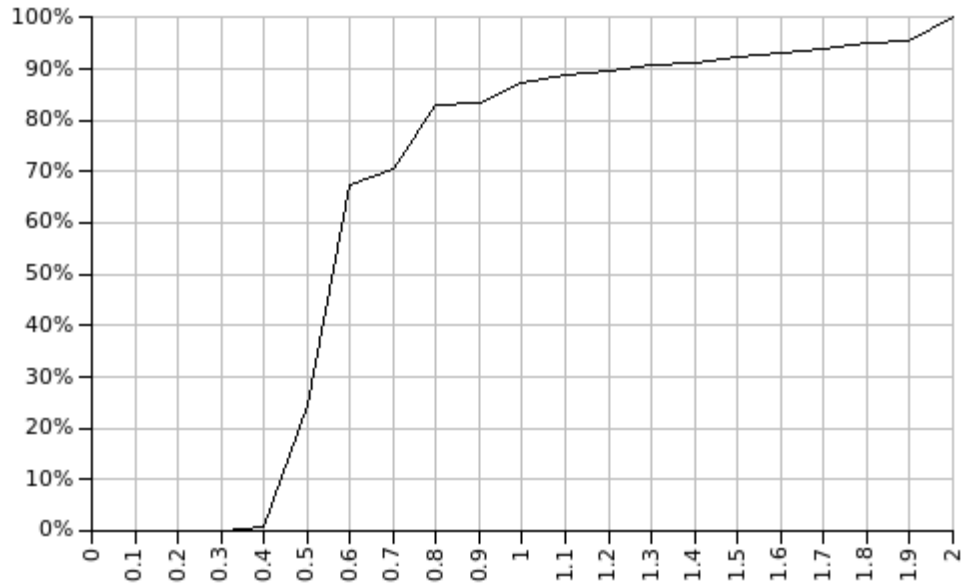


Illustration 8: The cumulative distribution function of $M_r(O.ngr)$; the 'n-grams' feature ($n = 3$). The labels on the value axis indicate the lower bounds of 100KB-sized memory bins, in megabytes. The

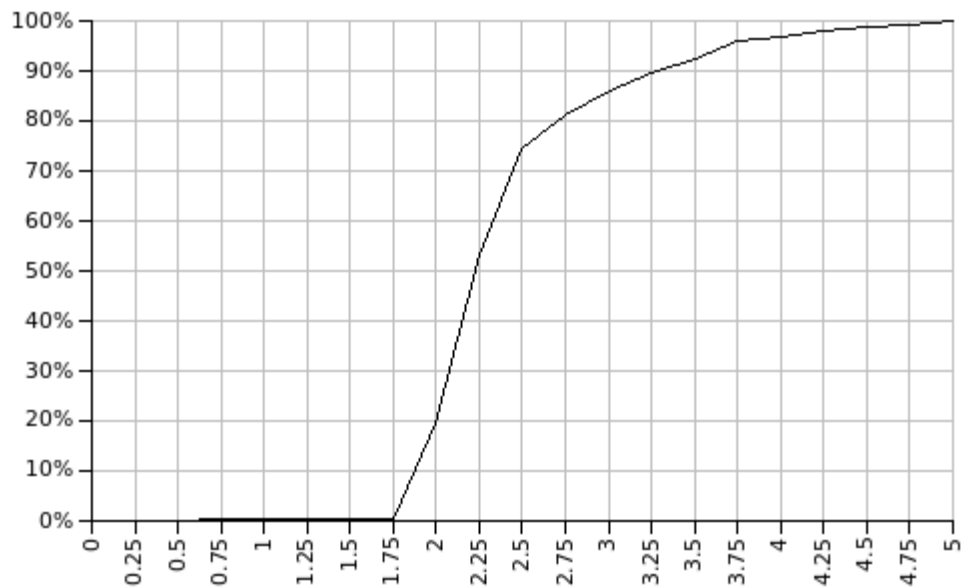


Illustration 9: The cumulative distribution function of $M_r(M.idx)$; the 'word index' feature (lemmatized, no parts-of-speech). The labels on the value axis indicate the lower bounds of 250KB-sized memory bins, in megabytes. The average is 2.61MB per message, the maximum is

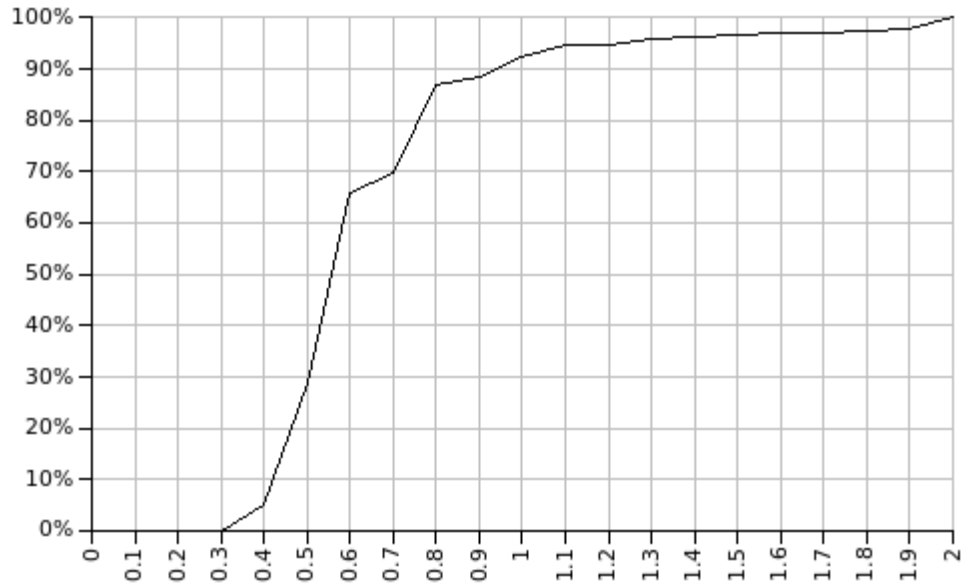


Illustration 10: The cumulative distribution function of $M_r(T.Ink)$; the 'links' feature. The labels on the value axis indicate the lower bounds of 100KB-sized memory bins, in megabytes. The average is 760KB

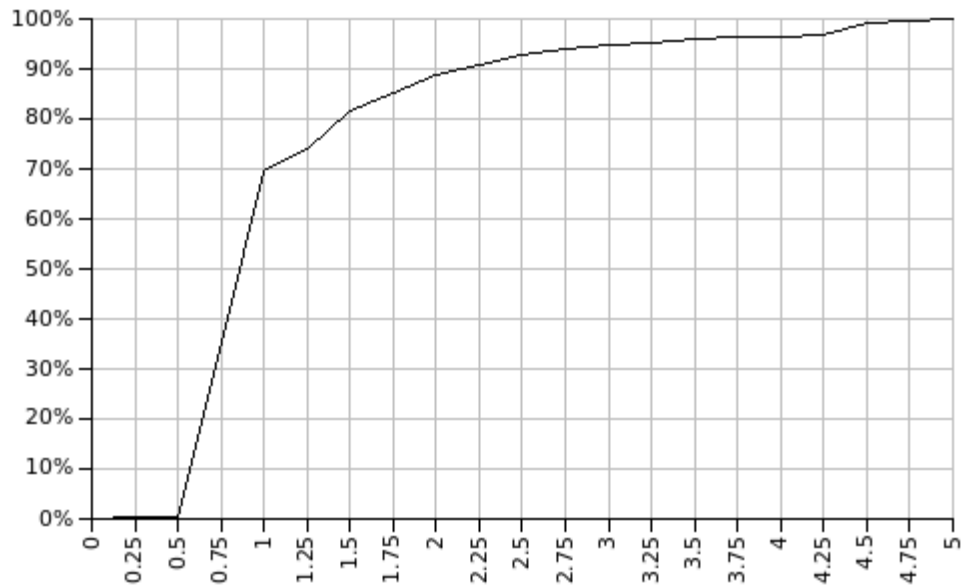


Illustration 11: The cumulative distribution function of $M_r(T.elm)$; the 'elements' feature. The labels on the value axis indicate the lower bounds of 250KB-sized memory bins, in megabytes. The average is

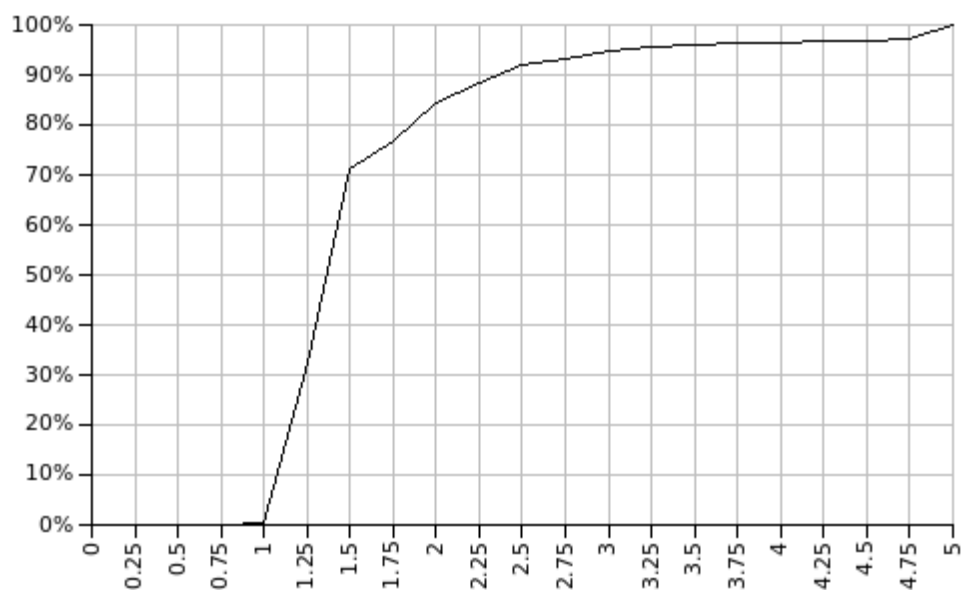


Illustration 12: The cumulative distribution function of $M_r(I.sal)$; the 'salting' feature. The labels on the value axis indicate the lower bounds of 250KB-sized memory bins, in megabytes. The average is