Microsoft Outlook 2003 Spam Filter: Under the hood

Table of contents:

There are 28 anti-spam add-ins for Outlook in our software archive MAPIStore.Com — those are real add-ins by third-party developers, fully integrated with Outlook and implementing the newest spam filtering methods, including self-training ones based on various modifications of the Bayesian method[1]. Microsoft Outlook 2003 is supplied with built-in junk mail filter based on "state-of-the-art technology developed by Microsoft Research[2]". This Microsoft technology is considered in our article.

Spam filtering is a very difficult task, which is evident at least from the fact that, despite numerous software solutions offered and attempts to enforce anti-spam legislation, our mailboxes are still stuffed with junk mail. And the problem is growing even more serious year after year.

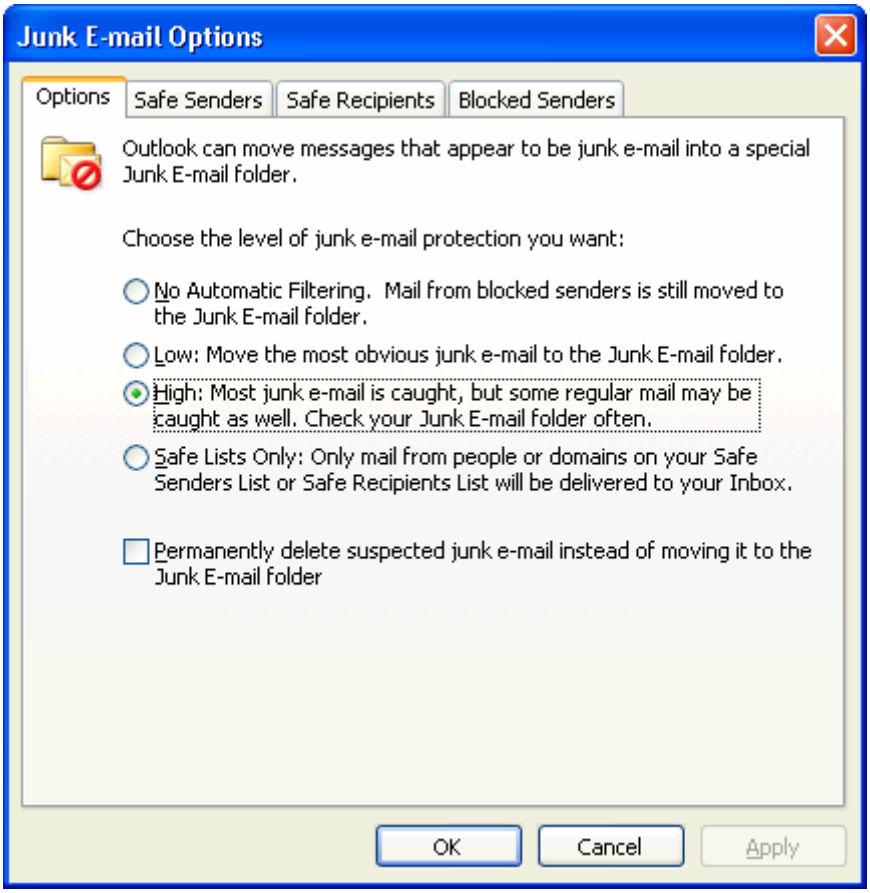
Probably, that's why we failed to learn about the new Microsoft technology from publications about Microsoft Office 2003 much more than one could read in the program help: "technology : that is used to evaluate whether an unread message should be treated as a junk e-mail message based on several factors, such as the time it was sent and the content of the message[3]". In our opinion, spam filtering has something in common with cryptography: the fact that the algorithm is kept in secret doesn't make it crack-proof, nor does it guarantee effective spam filtering. And that's why we decided to publish this article.

---

[1] The essence of the Bayesian's method is that the filter, based on the analysis of earlier received messages, predicts if the new message is spam or not. First, a user trains the filter by marking wanted and unwanted mail. During the training, filter's prediction accuracy is growing, and, as some of the developers claim, trained filter screens off 98-99% of junk mail, the rate of errors not exceeding 0.1%. For details, see links 1-3 in appendix B.

[2] Quotation from the article "About the Junk E-mail Filter" in Microsoft Outlook 2003 help system. See link 7 in appendix B.

[3] Ibidem



Our company is specializing in development of add-ins for Microsoft Outlook, therefore we thoroughly scrutinize all innovations in Microsoft Office — both as users and as software developers. As users, we were a bit surprised to find a hint at usage of some variant of the Bayesian's method for the filtering in one of the articles about Microsoft Office 2003 — and we expected to see something trainable. However, the filter appeared to us as kind of "black box" — we could only choose between the two filter response levels — low and high. No more controls were provided. We tried to train the filter, marking messages as spam and vice versa, but all in vain.

Then we started studying it as software developers. In Microsoft Outlook, unlike Outlook Express, there are advanced and well documented facilities for creation of add-ins, using which a software developer can add almost any feature to Outlook. Alas, here disappointment was awaiting us again: neither the filter itself, nor any other anti-spam components could be detected (senders black list etc.) anywhere. Of course, they did exist and they worked, but a third-party software developer couldn't get access to them or find out any information about them.

So, we had to go one level deeper in our investigation. The text below will abound in technical details, so a non-professional reader may skip to the final section and refer to the technical information where necessary. And software developers will find a couple of exercises we've prepared for them to check if they have learnt to detect junk mail with up-to-six-decimal-digits accuracy, as the filter does.

General information about the filter

The filter consists of two files: a data file and a file containing program component (a standard COM-component, which, unfortunately, has no type library). The files are located in the Office11 subdirectory of the directory with Microsoft Office:

Created	Size	File Name
07.07.2003 12:36	2058343	OUTLFLTR.DAT
08.07.2003 10:48	115288	OUTLFLTR.DLL

A program for Windows which will just display the line "Hello, world!", compiled with a modern compiler usually takes 40-50 K of the disk space, and this is a cause for never-ending jokes among software developers. Most of the program's code is taken by standard libraries attached by the compiler while assembling the program. The libraries are attached only once, so of 115 K is capable of something more than just display the line "Hello, world!" trice. Nevertheless, the size of the program file suggests that we shouldn't expect much from the filter.

OUTLFLTR.DAT File Format

99,99% of the file is taken by 93555 entries of the following format:

```
struct HashRec
{
   unsigned char  hash[16];
   WORD           reserved;
   float          weight;
};
```

The data file was created with the widely known algorithm MD5, which calculates 128-bit (or 16-byte) hash[4] based on source data block of unlimited size. The standard hash calculating algorithm looks as follows:

```
void md5(
   unsigned char* data,  // IN, data block
   int length,           // IN, size of data block
   unsigned char *digest // OUT, pointer to output buffer
   )
{
   MD5_CTX context;
   MD5Init(&context);
   MD5Update(&context,data,length);
   MD5Final(digest, &context);
}
```

[4] A hash-function calculates the value called "hash" based on input data block. The main feature of hash-function is that it has no inverse function: input data cannot be calculated based on hash. The only possible way to restore the source data is to sweep all probable source values. A good algorithm has no anomalies, so it doesn't allow any narrowing of the sweep area through hash analysis (for example, to estimate approximate size of the source data block).

First, the code MD5Init initializes the data structure to be used for hash calculation (in fact, it initializes four 32-bit values in the structure with some constant "magical" values). The code MD5Update updates those four values based on the counting of the input data block, and the code MD5Final completes calculating and returns the 128-bit hash value into the "digest" variable.

In our case, the algorithm was slightly modified:

```
void ms_md5(
   unsigned char* key,
   unsigned char* data,
   int length,
   unsigned char *digest
   )
{
   MD5_CTX context;
   MD5Init(&context);
   MD5Update(&context,key,16);
   MD5Update(&context,data,length);
   MD5Final(digest, &context);
}
```

An additional 16-byte-long key is transmitted into the function, which in fact "shift" initialization of the source values performed by the code MD5Init, so data block hash is calculated on a different base.

The file, as the reader probably has already guessed, contains an absolutely standard dictionary of "word — weight" pairs. The only special moment is that instead of a word there is its hash in a dictionary: Microsoft decided not to write the list of good, not so good, and just nasty words in the plain form to user's disk on installation of Microsoft Office.

However, the filter can be easily attacked on dictionary: it would be enough to calculate a million or two hashes for an English vocabulary, with all possible short strings like "r1" added (the procedure of calculating one or two millions will take 1-3 seconds on a modern computer), and to check if any of the calculated hashes are contained in the filter dictionary. Here is a fragment of the filer dictionary recovered by this method:

Entry #	Word	Weight	Hash
2133	63	-0.009834	538bd1b2ab04f2d7205a3a9dd4010528
2134	recurring	0.047126	15e284413ed5a3e15b5860e9e03e9a1d
2135	market	-0.007610	1f719136265c6f4448f5a0d6324c4eef
2136	specifically	0.017916	23c3c2ab9736a569f2058f07fea8a08d
2137	r1	0.009694	5bfa887cb32dca0ee89e22ca0d55eff7
2138	common	-0.061535	f43de2a694cedee516d616f3bf0a91e8
2139	coffee	0.001494	284b6e517a515b6e535061aff1ce9a8f
2140	adv	0.034531	64955dfd3566eff0318fd6c212c88825
2141	via	-0.049830	2b739a689e2474d8f34d2369e9390c6d
2142	thing	0.024440	fa18559520b85b4b654aba091783096b

Negative weight decreases the probability of treating a message as spam, while positive weight makes that more likely. So, words with negative weight may be considered "good", while words with positive weight may be considered "bad".

However, what was the purpose of "shifting" the standard hash? The answer is quite evident. A word will have different weight depending on where it appears: in the message body or in the subject field.

```
DWORD bodyKey[] = {0x0408c00d, 0x4794990f, 0x79c052ad, 0xe601c933};
DWORD subjKey[] = {0xda1d7c3b, 0x49eb6186, 0xba1f419e, 0x7267237e};
```

To calculate the total weight of a message, the filter calculates hash for each word with bodyKey as the key, and checks if the resulting hashes are contained in the data file, thus obtaining weights for words in the message body. To calculate the total weight of message subject, the subjKey is used as the key. Certainly, the same word will have different hashes if they are calculated on different offsets.

Therefore some words appear twice in the data file:

Entry #	Word	Weight	Context	Hash
55833	screensavers	-0.020422	body	dbd50355e5e865c5fc6d97cb4c8eaa28
60205	screensavers	0.065368	subject	716e204ac490ac3f26af4e49535cd5ea
3261	linux	0.001100	body	9796b4300be5b94e840969d695ba5797
85227	linux	-0.021932	subject	cd2be8031b4bfe90c5756100bbea3d0f
2	free	0.072806	body	1ae8d55aaeee41e162b11a8aa3681b2d
508	free	0.069711	subject	df6cdc35bc39daf2cb5e63b29fc8faad

Thanks to these tricks, the file format is very simple. And to keep it so, many of the constants used to calculate message weight are also stored in that format (will be discussed a bit later).

Finally, we would like to note that hashes in the filter dictionary are calculated by character string in lower case in the Unicode format; besides English words, the dictionary also contains words from other languages. Though, the proportion is not even: English is apparently dominating. You will find the dictionary we've restored in the appendix to this article.

Now, as we have analyzed the main contents of the file, we can formalize its format using syntax like in C language:

```
struct FileHeader
{
  DWORD signature = 0xadfbcade;
  DWORD unknown1[] = {1,2,32,16, 2099098, 29573672,29,1,64};
  DWORD hashRecSize = 2058210;     // = recNumber*sizeof(HashRec)
  DWORD recNumber = 93555;
  DWORD fileHeaderSize = 93;       // = sizeof(FileHeader)
  DWORD scoreMapSize = 40;         // = scoresNumber*sizeof(ScoreMap)
  DWORD scoresNumber = 10;
  DWORD scoreMapOffset = 2058303;
  double q1, q2, q3;
```

```
  char   unknown2[] = { 16,0,0,0,1};
},
struct HashRec[FileHeader.recNumber]
{
  unsigned char hash[16];
  WORD          reserved = 0x0101;
  float         weight;
},
struct ScoreMap[FileHeader.scoresNumber]
{
  float         score;
};
```

The file signature is used for its check on loading. The HashRec structure was considered in details above, and the ScoreMap structure as well as the variables q1, q2 and q3 will be considered later. Most of the unknownX variables contain additional size parameters for the structures and their inner elements, therefore we didn't try to find out the exact purpose of 41 unknown bytes. Many of the other elements obviously could have been omitted too as there is apparent redundancy for additional checks of the file integrity on loading.

Data file contents

We didn't set the object to recover the filter dictionary completely. However, by simple attack on the dictionary, we managed to recover about 68000 words for message body weight calculation and about 5500 words for message subject weight calculation — we used dictionaries only containing words with Latin letters and numbers, and even though there were, say, some German words, there were no words with national symbols (for example, "o" and "a" with umlauts). And we used no Korean, Japanese, or Chinese dictionaries, though the data file must have contained some Korean, Japanese, or Chinese. We tried a small Russian dictionary, and found several dozens of Russian words immediately.

So, 80% of the dictionary may be recovered very easily, which is quite natural: there is no reason for including rarely used words in the dictionary, as this will result in drastic result of the data file size.

However, analysis of the recovered words suggests that the filter dictionary was compiled in a fully automatic mode. First, it contains all HTML tags and all words somehow related to HTML (e.g.: 10px, 11px, 12px, ff0000 — font size, colors in the RGB format, etc.). Second, it contains a fair number of words which can hardly be called commonly used:

Entry #	Context	Word
73543	Body	visitorwantstochat
17316	Body	444553540000
16075	Body	**********
49061	Body	audiogalaxysatalite
87119	Body	nzmwmtawmteymduzotk
19959	Body	0050dac68030
71396	Body	woekfweoifeoiwfewfiwefoefokwe
20608	Body	riilldijgjgjg

87448	Body	20030507
29661	Body	20030424

Did Microsoft Research have a lot of different messages containing those words? Junk messages often contain senseless letter sequences such as "riilldijgjgjg". Some modifications of the Bayesian method, for example, when calculating message weight also count words not found in a dictionary, assigning them positive "anti-spam" weight. Besides, such words sometimes actually are unique message identifiers. Using such words for message filtering is senseless, as they are always different.

The supposition that the word riilldijgjgjg may have a double — a regular commonly used word with the same hash — must be swept away immediately. The probability that two words will have the same hash is $1/2^{128}$ — check how many words the largest dictionary known to you contains, and calculate the probability of finding two words with the same hash: divide the number of words by 2^{128}. If your dictionary contains a bit more than million words, the probability will be $1/2^{108}$.

However, the reader may ask quite reasonable question: if you used attack on dictionary, why does your dictionary contain such a strange word "riilldijgjgjg" ? The matter is that we also have spammed mailboxes, and we have tried the filter dictionary also on them. We received the spam message which contained that mysterious word (along with some more, no less mysterious) on December 18, 2002 (!).

Intrigued by that super-word, we decided to search for it in Google newsgroups (http://groups.google.com/) — and found some more messages with it in the anti-spam conference news.admin.net-abuse.sightings — junk mail samples from all over the world are sent to that conference.

We don't know how many of such 13-character words are there in the dictionary, and they can be only found by sweeping. However, it's quite possibly that 20% of the dictionary that remained unrecovered contain rather such garbage as riilldijgjgjg than words with national symbols.

After all, actually it doesn't matter, so we let specialists in the area deal with analysis of the dictionary quality. And for mathematicians we leave recovery of the algorithm used to assign weight to words. The purpose of this section was to give the reader overall idea of the dictionary contents.

State-of-the-art technology

Now it's time to consider the new technology itself. Less the details, the algorithm of message filtering is the following. The total weight for all words contained in the message body is calculated (if a message is in HTML format, the calculation is made both for text and HTML presentation and the results are summed up). Duplicate words are excluded; if the word is not found in the dictionary, it is assigned zero weight. Then, the total weight for words in the message subject is calculated. The result is added to the total weight of the message body. Further on, we will refer to the resulting amount as message weight.

Then, the filter performs eleven additional checks (including already mentioned check for message receipt time). If the message meets the check criterion, certain constant is added to its weight.

The resulting weight usually lies within the range from -10 to +10, where negative values means that the message is not spam, while positive ones mean it may be spam. After that, the message weight is converted into the range from 0 to 1 (normalized weight) using the formula:

```
x = ((weight-q1)*q2+q3)*log2e;
r = round(x); // rounding to the nearest integer
normalizedWeight = 1/(1+2(x-r)*2r);
```

The constants q1, q2, and q3 in this formula are constants stored in the data file (see the format description). The greater normalized weight is, the more likely the message may be spam.

The following table will help the reader to get the idea of how the formula works:

Weight	Normalized weight
-2.0	0.011867
-1.0	0.092953
0	0.466515
+1.0	0.881824
+2.0	0.984538

The ScoreMap table (see the file format description) looks as follows:

Index	Normalized weight
0	0.00000000
1	0.30000001
2	0.56000000
3	0.67100000
4	0.73000002
5	0.80000001

6	0.93099999
7	0.94999999
8	0.95999998
9	0.98000002

This table is used to assign the message index according to its normalized weight (for 0.85 the index is 5), which is the filtering result.

If spam protection level is set to LOW, messages with index exceeding 6 will be placed into the Junk Mail folder. If the protection is set to HIGH, messages with index exceeding 3 will go to the Junk Mail folder.

Now we are to sum up the procedure description:

1. The total weight of words in message body and subject is calculated
2. The weight is adjusted depending on the result of additional checks
3. The weight is converted into the range from 0 to 1 (normalized)
4. The normalized weight is converted into the range of 0..9 according to the table
5. The resulting value is compared with the chosen spam protection threshold
6. Based on the comparison result, the message is either placed into the Junk Mail folder or not.

We have already given rather thorough consideration to all the steps except additional checks. Now lets analyze some of those checks.

Message sending time check

Actually, this test involves several factors: message receiving day, the hour of the day when the message was received, and the time elapsed between the moments when the message was sent and when it was received.

The time between sending and receiving the message	Weight
Less than an hour	-0.188458
1 to 6 hours	0
6 to 24 hours	+0.083999
Over 24 hours	+0.104574

For example, weight of the word "porno" in the message body is 0.041693. So, if you permit yourself to check mail once a day just before leaving the office, messages sent to you in the morning will have such rating as if there is a couple of the that words in each. To all appearances, this test won't work at all with Microsoft Exchange Server: in this instance, the time of message arrival to the server is taken as receiving time, not the time when it reaches the end recipient; so the difference may only be meaningful if the message was unable to reach the server for a long time due to some conditions.

Message receiving day	Weight
Monday	-0.005292
Tuesday	-0.014362
Wednesday	-0.014959
Thursday	-0.004486
Friday	-0.003736
Saturday	+0.021732
Sunday	+0.021103

Don't forget that all these weights are simply added to the message weight. Let the reader guess himself why a message received on Saturday "most probably is spam", whereas a message received on Wednesday "must be not spam". Though the weight absolute values are quite small, so their influence on the resulting weight is also insignificant.

Hour	Weight	Hour	Weight	Hour	Weight
0	0.001160	8	0.003010	16	0
1	0.002731	9	0	17	-0.003565
2	0	10	0	18	0
3	0.002952	11	0.002243	19	0.001507
4	0.001054	12	-0.002727	20	-0.003430
5	0.004290	13	-0.007149	21	0
6	0.005275	14	-0.004555	22	0.003923
7	-0.001351	15	-0.007769	23	0.002025

The procedure of calculating the influence of message receiving hour is somewhat more complicated:

```
HourWeight = weight(hour) + weight(hour+1) + weight(hour-1)
```

Example: the message was sent on Thursday at 10:30 and received on Thursday at 19:15. Then, the test result will be the following:

```
0.083999 - 0.004486 + 0 + 0.001507 - 0.003430
```

Of note, the time taken for calculations is not local time (which is displayed by your mail client and which you can see in the right lower corner of the screen), it is system time, which can be different from local! For example, if you are in the time zone UTC+1 (Coordinated Universal Time, or GMT+1), the difference between local time and system time is one hour.

This strange fact looks like a mistake of software developers at Microsoft. It's clear that no matter where you are, in Moscow, London, or New York, the table of weights should remain the same! But with mail messages all dates are stored not in local time but in system. And someone was merely too lazy to get the local time offset and adjust the time.

Check of the Message Subject for Words in Uppercase

This test calculates the proportion of words with all letters in uppercase to all words in the message subject. If words in uppercase make 25% or more of the total number of words in the message subject, -0.015324 will be added to the message weight. Why the fact that message heading contains a lot of uppercase words reduces the probability of treating the message as junk mail instead of increasing it that's an enigma. Maybe, this is one more mistake of Microsoft?

Check of the Sign Number in the Message Subject

This test calculates the ratio of signs (symbols which are neither letters nor numbers) to the number of signs, letters, and numbers. If the ratio exceeds 8%, -0.011104 is added to the message weight. Here we have the same enigma as in the previous test: if the message subject is "C*H*E*A*P***V*I*A*G*R*A", why is the probability of treating it as spam lower? We have checked this moment several times, but it is really so!

Check of Duplicate Character Number

This test counts the maximum number of duplicate characters in the message subject (for the string "HELLO, ALL!" this number will be 2).

The number of duplicates	Weight
No duplicates	-0.135528
Two duplicates (like in the example above)	-0.119723
Three or four	-0.040340
5 to 8	0.055501
9 to 16	0.027391
17 to 32	0.056192
33 to 64	0.093917
65 and more	0.057513

The relevant weight from the table above is added to the total message weight. In many spam messages there is a sequence of meaningless (for you and us) symbols separated by several dozens of spaces. The test appears to be designed to screen off such messages.

The Rest Eight Checks

We decided not to scrutinize the rest checks thoroughly: they appear to be no more intellectual than those described above. Moreover, they didn't anyhow affect the weight of messages we processed with the filter. Whereas the checks analyzed above worked rather frequently.

Exercises for developers

We have prepared several exercises for those who would like to check their understanding of the facts described in the article. You can calculate weights of the given messages yourself, using the dictionary provided in the Appendix. Calculation of the message body weight is a rather hard task, as the dictionary, besides words, contains punctuation marks (full stops, commas, etc.) which also should be taken into account. All the messages given here are in the text format, they were randomly chosen from our mailboxes. The local time is two hours ahead of the system time.

Exercise 1.

```
Sent: Wednesday, 05.11.2003 17:25
Received: Wednesday, 05.11.2003 17:25
Subject: did you lose your keys

Do you "misplace" your keys a lot?

http://www.webrx123.com/misc/ekey.htm
```

Result:

Parameter	Value
Message subject weight	0.193284
Message body weight	0.923498
Analysis of the receiving time	-0.215740
Check for uppercase words in the subject	0
Check for the number of signs in the subject	0
Check for duplicate symbols	-0.135528
The total weight of the message	0.765514
The normalized weight	0.818629
Normalized weight index	5

Exercise 2.

Sent: Wednesday, 05.11.2003 17:39
Received: Wednesday, 05.11.2003 17:39
Subject: HOT SWEET YOUNG FREE BABES!

free pass to the best sites on the planet!
http://www.111pass.com/pass2/

Result:

Parameter	Value
Message subject weight	0.277984
Message body weight	1.080657
Analysis of the receiving time	-0.215740
Check for uppercase words in the subject	-0.015324
Check for the number of signs in the subject	0
Check for duplicate symbols	-0.119723
The total weight of the message	1.007854
The normalized weight	0.883568
Normalized weight index	5

Under the hood

As we have already noted in the beginning of the article, spam filtering is a quite difficult task. The vaunted "state-of-the-art Microsoft technology" is actually nothing more than a dozen of simple checks based on the quite evident ideas, such as "good messages cannot contain 20 spaces in the subject field, while with spam this is quite common" (spammers often add identifier in the message subject field, separated from the main text of the subject by several dozen spaces); or "a message received during the office hours most probably is not spam, while a message received at night or on weekend is likely to be spam". Moreover, software developers at Microsoft contrived to make a mistake in that check of message receiving time. All those checks can hardly be called "state-of-the-art technology".

The technology used for message content analysis is also far from being perfect. Microsoft has created a dictionary of several tens thousand words, and assigned different weights to the words in the dictionary. The message content analysis is nothing more than mere summation of weights of words contained in the message.

The worst thing is the fact that a user has no opportunity to train, modify, or disable all those dictionaries and checks! If the terms from your professional area were included into the Microsoft dictionary with "spam" weight, your business correspondence has a good chance of getting in Junk Mail, and the only thing you can do is to disable the filter. One more important weakness is the fact that the filter work in a similar way with all users (there are no personal dictionaries), so, having trained on his own mailbox, a spammer will easily get round filters of other Outlook users. And the third weakness: the filter is fine-tuned to deal with messages in English, so it will be much less effective, say, in filtering junk mail in Russian.

Thus, if your mailboxes are not stuffed with spam, you'd better hold back from using the filter offered by Microsoft than to rely on it. It's better to look through a few promo messages a day than to screen off an important business letter once. And if your mailboxes are swamped with spam, the Microsoft filter won't be a great help to you. Yes, it will be able of detecting a half of junk mail, but if you have 200 spam messages a day and the filter reduces this number to 100, would that solution be sufficient?

Some of Outlook 2003 users we talked to came to the same conclusion without knowing the technical details about the filter operation. Though, there also were some positive reports: "25 spam messages arrived to my mailbox today, 15 of them were caught by the filter, and I didn't notice any erratic responses. I didn't pay any extra money for it, so I'm quite happy with it".

However, this is a well-known tactic of Microsoft: they start with an obviously weak product (though, releasing a masterpiece with the version number 1.0 is well within the capabilities of such a powerful corporation) just to demonstrate their interest to a certain niche, but in a short time their product becomes #1 in terms of quality and other features, leaving the competitors far behind.

Well, what are the most probable developments on the market of anti-spam products for Outlook and what the user can expect? Microsoft may keep playing secrets with developers, gradually improving its filter. The niche of third-party products will be reducing then: many users will be unwilling to install an excellent filter in addition to a rather good one, provided there is no integration between them (for example, imagine there are two items in the context menu: "Mark as junk mail" and "mark as junk mail for 3rd Party Super Filter").

The user apparently won't benefit from such development of the situation. Microsoft also won't benefit from it — availability of third-party anti-spam add-ins will hardly be a threat to Outlook. At the same time, creating a super-filter as a unique feature of Microsoft Outlook to give the product a competitive advantage is impossible even for Microsoft.

Another variant: Microsoft may just disclose its existing interfaces, and dozens of developers will be able to adjust their solutions for Outlook 2003. Availability of open program interfaces will also make creation of new products for Outlook by third-party developers much easier.

We don't know yet what way Microsoft will opt for. However, some actions are to be taken within the next few months. By the time Outlook 2003 starts dominating over the previous versions, guides like "100 ways to get round the Outlook filter" will be in wide circulation among the spammers. Therefore Microsoft will have to improve its filter in the nearest future. Well, we shall see what we shall see.

In conclusion, we would like to answer some questions the reader may ask upon reading the article.

Did we disclose all the Microsoft secrets to spammers, thus having plunged the world into the abyss of spam?

Spammers don't need to know the details of filter performance: they can just send a message to their own copy of Outlook to see whether the filter will catch it or not. We admit that this article will save a spammer 5 minutes in finding the way to get through the filter. But on a larger scale this publication doesn't change anything. Spammers have known that junk mail filter don't like the word "porno" well before.

Which junk mail filter is the best?

We didn't run a comparative test of junk mail filters. Though we suppose a number of good articles and reviews must have been published over the Internet in the recent years.

Are you developing a junk mail filter of your own?

No, and we aren't going to so far. We are implementing one of the anti-spam technologies in our product Mail Storage Guard for Microsoft Exchange Server. However, our technology is quite different form one described here; Mail Storage Guard version with that technology will be available in early 2004.

Is it possible to gain understanding of the "hidden" Outlook interfaces and replace the filter with a different, better one?

It is possible to make the interface out and to replace the component itself. However, besides the component described in this article, most probably it will be also necessary to find interfaces to other components, such as "sender black list", say, to provide for dynamic editing, otherwise good performance can hardly be achieved. We don't know how complicated this task will be. After all, thorough consideration is necessary before attempting at doing that.

May I use the information from this article?

Appendix A: OUTFLTR.DAT file dump

For the full explanation of data file see the article "Microsoft Outlook 2003 Spam Filter: Under The Hood". Dump of the data file is divided into 10 parts.

Ctx	Count	Context description
S	5434	Weight for the token in the subject of email
B	68877	Weight for the token in the body of email
W	7	Day of week, receiving time check
H	18	Hour, receiving time check
T	3	Difference with sent time, receiving time check
D	8	Check for duplicated characters in subject
I	1	Check for signs in subject
L	1	Check for words in upper case in subject
U	19206	Weight of unknown token
	93555	**Total number of records in file**

File	File size	Tokens	Unknown
voc-01.html	1251414	1..10000	1270
voc-02.html	1268694	10001..20000	1717
voc-03.html	1270916	20001..30000	1869
voc-04.html	1272670	30001..40000	2059
voc-05.html	1271811	40001..50000	2145
voc-06.html	1273660	50001..60000	2380
voc-07.html	1270941	60001..70000	2313
voc-08.html	1269480	70001..80000	2076
voc-09.html	1267790	80001..90000	2077
voc-10.html	455485	90001..93555	1300
Total	**11872861**	**93555**	**19206**

Appendix B: Useful web links

1. **"A plan for spam"** by Paul Graham / Aug 2002

   This article describes the spam-filtering techniques used in the new spamproof web-based mail reader.

2. **"Better Bayesian Filtering"** by Paul Graham / Jan 2003

   This article was given as a talk at the 2003 Spam Conference. It describes the work author's have done to improve the performance of the algorithm described in A Plan for Spam, and what author plans to do in the future.

3. **"Spam Detection"** by Gary Robinson / Last updated at Oct 10, 2003

   This article describes another Bayesian technique and includes formulas and useful links.

4. **"How Outlook 2003 takes care of spam on its own"** by David Coursey, ZD Net / Oct 20, 2003

5. **"Microsoft applies its might to fighting spam"** by David Coursey, ZD Net / Oct 28, 2003
   http://comment.zdnet.co.uk/davidcoursey/0,39020667,39117445,00.htm

6. **"Spam Slayer: New Antispam Weaponry"** by Tom Spring, PCWorld.com / Oct 27, 2003
   http://www.pcworld.com/news/article/0,aid,113104,00.asp

7. **"Help Prevent Junk E-Mail Messages with Outlook 2003"** by Microsoft / Apr 28, 2003
   http://www.microsoft.com/office/editions/prodinfo/junkmail.mspx

8. **"Skirmishing With Spam"** by Maryfran Johnson, ComputerWorld / Aug 4, 2003

   This article describe the testing of open-source spam filter SpamBayes for Microsoft Outlook