MAY 2005

virus BULLETIN

The International Publication on Computer Virus Prevention, Recognition and Removal

CONTENTS

ISSN 0956-9979

IN THIS ISSUE

HAPPY 'B-DAY'

The beginning of March 2005 saw 'B-day': another rash of Bagle variants, with at least five new variants appearing during the course of one day. Gabor Szappanos describes one of the new brood: Bagle.BI.

page 4

EVOLUTIONARY TALES

As an unusual example of self-compiling malware and a novel misapplication of artificial intelligence, W32/Zellome is an interesting specimen. Peter Ferrie and Heather Shannon have the fascinating details – including the reason why Zellome is heading for extinction.

page 7

IN THE SPOTLIGHT

More than two years have passed since *VB*'s last review of *AhnLab*'s *V3Net*, so Matt Ham decided it was about time to put it through its paces once again and find out how the product has matured.

page 16

vbSpam supplement

This month: anti-spam news & events; challenges in spam filter evaluation.

'If the AV industry is slow to respond to attackers and their new motives and techniques, it will fail to protect its customers.'

Ken Dunham, iDEFENSE Inc.

AGGRAVATING ADWARE AND SINISTER SPYWARE

Adware and spyware have become a serious problem in the last two years. The anti-virus industry has been slow to respond, creating a new niche for security products, such as *Ad-Aware* and *Spybot Search and Destroy*, to fill the gap.

According to various end-user licence agreements and software distributions, adware and spyware can be legal. However, many corporations and individuals have found that these programs are unwanted and that they are difficult to identify and remove. Just because it is legal, it is not necessarily ethical or proper to allow such installations of code. Today, the average system administrator faces a barrage of these threats without any solid technical solutions to protect a large network.

In 2002, *iDEFENSE* identified 333 adware/spyware threats. That number rose to 1,304 in 2003 and to 8,804 in 2004. While individuals can fight off the most common threats with anti-adware/spyware utilities, corporations with extensive computer networks are left wondering how best to tackle the problem.

An informal survey administered through the AVIEN forum and the administrators of various large networks revealed that the anti-adware/spyware software solutions available today are significantly lacking the features that large corporations require. While recent products have attempted to address the problem, administrators do not feel that they can trust these new products or that they have the necessary integration and control measures to be able to run them on their networks in conjunction with required anti-virus efforts.

Even if an integrated, centralized, trusted solution emerged today, it would take time for organizations to implement it. Meanwhile, adware and spyware runs rampant on corporate machines worldwide. A recent DNS cache poisoning incident proves this point.

In February 2005, attackers hacked into multiple servers, poisoning the cache of DNS servers to which they had gained unauthorized access. This spread to lower-level servers within the DNS network to poison other servers. Attackers then manipulated various domains and IP addresses to host hostile code, which was mostly adware and spyware, and in one case, a backdoor Trojan.

This incident impacted thousands of computers and large networks globally. *Internet Explorer* users vulnerable to either MS04-013 or MS05-002 were infected silently with more than 17 MB of code. This code contained more than 45 different variants of code within 17 different families. If the end user clicked on a few dialog boxes that appeared during various installations following the attack, even more adware and spyware was downloaded on the computer.

The DNS cache poisoning incident was a highly organized, sophisticated, multi-stage attack designed for financial criminal gain. Multiple trends in both the commercial and malicious code worlds have demonstrated that money is now the motive for many hackers. This indicates that more sophisticated and organized criminal attacks will take place in 2005. If the anti-virus industry is slow to respond to attackers and their new motives and techniques, it will fail to protect its customers.

iDEFENSE carried out a basic test for adware/spyware detection. We scanned very common adware and spyware with 13 updated anti-virus programs, *Ad-Aware*, and *Spybot Search and Destroy*. The results: *Ad-Aware* detected 100 per cent, *Spybot Search and Destroy* 92 per cent, and on average, anti-virus programs detected only 31 per cent of the threats (ranging from 0–67 per cent).

With money as the motive behind adware and spyware, and some less than reputable companies on the scene, we expect the problem to worsen in 2005. The anti-virus industry needs to respond rapidly to new threats and provide solid, integrated, centralized solutions to new problems as they emerge.

NEWS

THREE (DAYS) IS THE MAGIC NUMBER

Virus Bulletin is very pleased to announce that this year the *VB* conference will run in a three-day format, with the VB2005 conference programme beginning at 14.00 on Wednesday 5 October. The change has been made as a result of feedback from VB2004 and a record number of submissions for this year's conference. VB2005 in Dublin will feature a truly packed programme, with an outstanding line-up that includes additional papers in all tracks – Corporate AV, Technical AV and both corporate and technical spam. The full programme can be found at http://www.virusbtn.com/conference/.

2.594.00 IS NOT THE MAGIC NUMBER

A red-faced *Trend Micro* has apologised to its customers for the release of a faulty update file that caused chaos for thousands of computer users worldwide.

Official Pattern Release 2.594.00, released in the morning of Saturday 23 April (Japanese time), caused 100% CPU usage, system slow down, and in some cases complete system failure on machines running *Windows XP SP 2* and *Windows 2003 Server*. Although *Trend* staff removed the file from the Active Update list just 90 minutes later, the company estimated that it had already been downloaded between 300,000 and 350,000 times and support staff are reported to have received in the region of 370,000 calls about the problem. The fault was blamed on insufficient testing of the pattern file in the rush to add detection of the Rbot family of worms. To add to its woes, *Trend*'s share price took a knock, falling 4.7 per cent, and it is expected that the incident will have an adverse effect on the company's second quarter results.

US EARLY WARNING CENTRE OPENS

The Cyber Incident Detection Data Analysis Center (CIDDAC) has announced the opening of its National Operations Center at the University of Pennsylvania. The non-profit organization aims to set up an automated real-time threat reporting system, which will enable it to issue early warning alerts to its member companies and provide information to law enforcement agencies. The Center will gather data from its member companies via a network of intrusion monitoring machines known as Real-time Cyber Attack Detection Sensors. As soon as a threat is detected CIDDAC will issue an alert to member organizations and law enforcement agencies. Crucially, the identity of the reporting companies will remain confidential. Enterprise-level membership of CIDDAC is priced at $10,000 per annum. See http://www.ciddac.org/.

Prevalence Table – March 2005

Virus	Type	Incidents	Reports
Win32/Netsky	File	35,968	59.88%
Win32/Sober	File	7,861	13.09%
Win32/Bagz	File	7,167	11.93%
Win32/Bagle	File	6,006	10.00%
Win32/Zafi	File	802	1.34%
Win32/Funlove	File	504	0.84%
Win32/Mydoom	File	427	0.71%
Win32/Dumaru	File	372	0.62%
Win32/Mabutu	File	227	0.38%
Win32/Lovgate	File	157	0.26%
Win32/Valla	File	114	0.19%
Win32/Klez	File	103	0.17%
Win32/Bugbear	File	81	0.13%
Win32/Pate	File	57	0.09%
Win95/Tenrobot	File	39	0.06%
Win32/MyWife	File	33	0.05%
Win95/Spaces	File	25	0.04%
Win32/Mytob	File	19	0.03%
Win32/Elkern	File	17	0.03%
Win32/Swen	File	14	0.02%
Win32/Myfip	File	13	0.02%
Psyme	Script	10	0.02%
Hiac	Macro	7	0.01%
Lunch	Macro	6	0.01%
Class	Macro	4	0.01%
Win32/Ganda	File	4	0.01%
Cap	Macro	3	0.00%
IEStart	Script	3	0.00%
Hope	Macro	2	0.00%
Redlof	Script	2	0.00%
Win32/Kriz	File	2	0.00%
Win32/Magistr	File	2	0.00%
Others[1]		19	0.03%
Total		60,070	100%

[1]The Prevalence Table includes a total of 19 reports across 15 further viruses. Readers are reminded that a complete listing is posted at http://www.virusbtn.com/Prevalence/.

VIRUS ANALYSIS 1

IN LIMITED DISTRIBUTION ONLY

Gabor Szappanos
VirusBuster, Hungary

March 2005 started with 'B-day': a new rash of Bagle variants. At least five different variants appeared during the course of one day, with a couple of hours elapsing between the individual bursts of new variants. This continued until a website that was hosting the email addresses used by the worm was shut down. This website played a crucial role in the worm's infection cycle, its removal effectively stopping the worm from spreading.

Early samples of (the dropper part of) the worm were observed in unencrypted form, but later variants were packed with a modified version of the *PeX* exepacker (as this was available in source code, it would not have been difficult to modify). The second stage unpacker code was unchanged, but the first stage code (which unpacks the second stage) had been modified slightly in order to avoid detection by some anti-virus scanners.

This article describes one of the specimens of this new batch: Bagle.BI. The other variants are very similar – in fact, the dropper-downloader components are basically the same (with the exception of the first unpacked one), only repacked to bypass detection. The worm component varies slightly, but in most cases it is only the string constants that have been changed.

THE WORM

The worm is a 29,700-byte executable. It listens on port 80 for incoming connections.

On execution, the worm part runs through Bagle's usual anti-Netsky routine. First it creates a series of Netsky-related mutexes to prevent them from running:

```
MuXxXxTENYKSDesignedAsTheFollowerOfSkynet-D
'D'r'o'p'p'e'd'S'k'y'N'e't'
_-oOaxX|-+S+-+k+-+y+-+N+-+e+-+t+-|XxKOo-_
[SkyNet.cz]SystemsMutex
AdmSkynetJklS003
____->>>>U<<<<-____
_-oO]xX|-S-k-y-N-e-t-|Xx[Oo-_
```

Then it attempts to delete several Netsky-related registry startup keys from both HKCU and HKLM:

My AV	Jammer2nd
Zone Labs Client Ex	FirewallSvr
9XHtProtect	MsInfo'
Antivirus	SysMonXP
Special Firewall Service	EasyAV
service	PandaAVEngine
Tiny AV	Norton Antivirus AV
ICQNet	KasperskyAVEng
HtProtect	SkynetsRevenge
NetDy	ICQ Net

However, when doing this, the worm uses an incorrect registry location:

SOFTWARE\Microsoft\Windows\CurrentVersion\Ru1n

As a result, this counterattack is unsuccessful.

Then the worm installs itself on the system and registers itself for startup in the (bogus) 'HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Ru1n' key. The worm checks whether it was started with the '-upd' switch – if so, it terminates the windlhhl.exe process. If it is not running from the Windows system directory as windlhhl.exe, the worm copies itself there, executes this copy and terminates the original program.

Then the worm checks whether smtp.earthlink.net is accessible. If it is not, it removes the registry key 'HKCU\SOFTWARE\ertrtt', and exits. It also exits at this point if the worm is executed after 10 August 2006.

Next, a dummy download thread is started. The code is the same as in the downloader component, except that the download list contains only the dummy location 'http://localhost/script1.php'. Since this code is repeated in the downloader component, it is redundant here. The fact that this code was not removed from the worm component suggests that this compound was developed from reused code parts, in a hurry, without careful design.

Finally, the worm enters an endless loop. In this loop it restores its (bogus) registry startup key, and performs the address download routine.

The worm downloads the file 'http://oceancareers.com/z/sss2.php' (after purging the URL cache), saves it into the Windows directory as EML.EXE and extracts the email addresses from its content. The extract routine is almost the same as in Bagle.BI's spam tool (which is described later), except that the upper limit for the email addresses is 450 characters here, rather than 500.

The address is invalidated if it contains the string 'D3' or 'd3', or one of the following:

@eerswqe	help@	linux	samples
@derewrdgrs	info@	listserv	abuse
@microsoft	nobody@	certific	panda
rating@	noone@	sopho	cafee
f-secur	kasp	@foo	spam
news	admin	@iana	pgp
update	icrosoft	free-av	@avp.

anyone@	support	@messagelab	noreply
bugs@	ntivi	winzip	local
contract@	unix	google	root@
feste	bsd	winrar	postmaster@
gold-certs@			

The messages are sent out in different threads, with a maximum of 15 threads.

The message body is contained within <html><body> and </body></html> tags, and could be one of the following:

- `new price<br><br>`
- `price<br><br>`
- `<br>The password is <img src="cid: "><br>`
- `<br>Password: <img src="cid: "><br>`

The message attachment is the ZIP-packed dropper component. The name of the ZIP archive is one of the following:

price	price_new	08_price	new_price
price2	price_08	newprice	new__price

The name of the dropper within the ZIP archive is doc_01.exe.

THE DROPPER

The 34,304-byte dropper part of Bagle.BI creates a copy of itself in the Windows System directory as WINSHOST.EXE. Then it creates startup keys in the registry:

Shell_TrayWnd=winshost.exe

Under two locations:

HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

Next, it drops the 18,944-byte WIWSHOST.EXE (the downloader component) into the Windows System folder.

Finally, it finds the *Explorer* process running in the memory, writes the downloader component in the *Explorer* address space, and starts a new thread on this copy. This way, even if running a firewall, the user is fooled into believing that the *Windows Explorer* starts the Internet communication, and tends to allow it. This method is applied if the operating system is at least *Windows 2000* (i.e. the major version is at least 5). Otherwise, the dropped downloader is loaded as a DLL, and executed this way.

THE DOWNLOADER

The downloader part of Bagle.BI starts by modifying the HOSTS file, overwriting its original content to block more than 120 anti-virus-related host names, thus disabling access to anti-virus definition update sites (a full list can be found at http://www.virusbtn.com/articles/virusbulletin/analysis/2005/05_4-7.xml).

Once it has done that the downloader stops 110 services that belong mostly to anti-virus and firewall programs (the full list can be found at http://www.virusbtn.com/articles/virusbulletin/analysis/2005/05_4-7.xml). This method is applied if the operating system is at least *Windows 2000* (i.e. the major version is at least 5).

Next the Trojan starts a thread which, in an endless loop, removes the registry entries of some common anti-virus programs, thus preventing their successful installation. It deletes the following keys from HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run:

Symantec NetDriver Monitor	APVXDWIN
ccApp	KAV50
NAV CfgWiz	avg7_cc
SSC_UserPrompt	avg7_emc
McAfee Guardian	Zone Labs Client
McAfee.InstantUpdate.Monitor	

The following registry keys are removed entirely from HKLM\SOFTWARE:

Symantec	KasperskyLab	Panda Software
McAfee	Agnitum	Zone Labs

Another thread is started, which scans all fixed drives recursively, searching for two files from two lists. The files on the first list are deleted, while the files from the second list are renamed.

The first file list seems to be unfinished, since it consists of only one filename: mysuperprog.exe. It is likely that this list is being kept for further development.

According to the second list the following renaming of files occurs:

CCSETMGR.EXE	->	C1CSETMGR.EXE
CCEVTMGR.EXE	->	CC1EVTMGR.EXE
NAVAPSVC.EXE	->	NAV1APSVC.EXE
NPFMNTOR.EXE	->	NPFM1NTOR.EXE
symlcsvc.exe	->	s1ymlcsvc.exe
SPBBCSvc.exe	->	SP1BBCSvc.exe
SNDSrvc.exe	->	SND1Srvc.exe
ccApp.exe	->	ccA1pp.exe
ccl30.dll	->	cc1l30.dll
ccvrtrst.dll	->	ccv1rtrst.dll
LUALL.EXE	->	LUAL1L.EXE
AUPDATE.EXE	->	AUPD1ATE.EXE
Luupdate.exe	->	Luup1date.exe

LUINSDLL.DLL	->	LUI1NSDLL.DLL
RuLaunch.exe	->	RuLa1unch.exe
CMGrdian.exe	->	CM1Grdian.exe
Mcshield.exe	->	Mcsh1ield.exe
outpost.exe	->	outp1ost.exe
Avconsol.exe	->	Avc1onsol.exe
Vshwin32.exe	->	Vshw1in32.exe
VsStat.exe	->	Vs1Stat.exe
Avsynmgr.exe	->	Av1synmgr.exe
kavmm.exe	->	kav12mm.exe
Up2Date.exe	->	Up222Date.exe
KAV.exe	->	K2A2V.exe
avgcc.exe	->	avgc3c.exe
avgemc.exe	->	avg23emc.exe
zonealarm.exe	->	zonealarm.exe
zatutor.exe	->	zatutor.exe
zlavscan.dll	->	zlavscan.dll
zlclient.exe	->	zo3nealarm.exe
isafe.exe	->	zatu6tor.exe
cafix.exe	->	zl5avscan.dll
vsvault.dll	->	zlcli6ent.exe
av.dll	->	is5a6fe.exe
vetredir.dll	->	c6a5fix.exe

Note that the end of the list does not make sense. It seems that the source and destination lists are updated separately, without considering their correlation.

Next, the Trojan stops two other services, wscsvc and SharedAccess. However, the latter has already been included in the earlier list of stopped services.

The same procedure is included in the Trojan code again, which is called twice in a row, to stop the processes separately. Moreover, in order to do this, the necessary functions from ADVAPI32.DLL are imported again, despite the fact that they are already available from the earlier stop process loop. Additional unused procedures are also left in the Trojan code.

Afterwards the downloader stops the following (mostly anti-virus-related) processes in a separate thread in an endless loop:

AVXQUAR.EXE	AUTOTRACE.EXE
ESCANHNT.EXE	AUTOUPDATE.EXE
UPGRADER.EXE	FIREWALL.EXE
AVXQUAR.EXE	ATUPDATER.EXE
AVWUPD32.EXE	LUALL.EXE
AVPUPD.EXE	DRWEBUPW.EXE
CFIAUDIT.EXE	AUTODOWN.EXE
UPDATE.EXE	NUPGRADE.EXE
NUPGRADE.EXE	OUTPOST.EXE
MCUPDATE.EXE	ICSSUPPNT.EXE
ATUPDATER.EXE	ICSUPP95.EXE
AUPDATE.EXE	ESCANH95.EXE

Finally, the downloader thread starts, which, in an endless loop (waiting six hours between each full loop) attempts to download the next component from 154 web locations (a full list of the web locations can be found at http://www.virusbtn.com/articles/virusbulletin/analysis/2005/05_4-7.xml).

The downloaded file is saved in the Windows directory as _re_file.exe, and is executed. Before each download the URL cache is cleared in order to fetch the latest content on the websites – in case one of the files has been updated.

The last four bytes of the downloaded file serve as a checksum using a CRC32 algorithm. When the download is complete, the checksum is compared with the one calculated from the downloaded file. If the two do not match, the downloaded file will not be executed.

THE SPAM TOOL

The websites used by these Bagle variants were mostly empty. However, they were used actively during 'B-day' to seed the new variants. Only two of the sites showed any significant content.

Originally, 'http://www.astergut.at/zo2.jpg' carried a simple downloader, which attempted to download 'http://www.comtec1.de/pr22222.jpg'. The latter was a spam tool, but it was soon removed. Then the spam tool was moved to take the place of the downloader.

Another site, 'http://www.yamdiamonds.com/zo2.jpg', became active later, carrying the same spam tool – which was probably one of the main reasons behind the Bagle hysteria. All the trouble a poor virus writer has to go through to get a good list of email addresses!

Both the downloader and the spam tool are equipped properly with the checksum at the end, so they are very likely to be the creations of the Bagle gang.

The spam tool collects addresses and sends them to a website. In order to run only once on an infected system, it marks its presence by creating a registry key, 'HKCU\Software\zseewr', with value 'zseewr'. If this is found, the hard drive scanning routine will be skipped.

Then the tool searches all local drives for email addresses and looks for files with extensions .wab, .txt, .msg, .htm, .shtm, .stm, .xml, .dbx, .mbx, .mdx, .eml, .nch, .mmf, .ods,

.cfg, .asp, .php, .pl, .wsh, .adb, .tbb, .sht, .xls, .oft, .uin, .cgi, .mht, .dhtm and .jsp.

From these files, it grabs anything similar to email addresses (i.e. strings containing the character '@' and which are longer than five bytes and shorter than 500 bytes).

Then a maximum of 50 threads are started that validate the collected addresses. First the tool checks whether the domain part of the address can be resolved, then it checks whether it is a forbidden or invalid address by searching the following strings: '@.', '.@', '..', 'rating@', 'f-secur', 'mts@lebanon-online.com.lb', 'news', 'update', 'anyone@', 'bugs@', 'contract@', 'feste', 'gold-certs@', 'help@', 'info@', 'nobody@', 'noone@', 'kasp', 'admin', 'icrosoft', 'support', 'ntivi', 'unix', 'bsd', 'linux', 'listserv', 'certific', 'sopho', '@foo', '@iana', 'free-av', '@messagelab', 'winzip', 'google', 'winrar', 'samples', 'abuse', 'panda', 'cafee', 'spam', 'pgp', '@avp.', 'noreply', 'local', 'root@', 'postmaster@'.

Validated addresses are copied in encrypted form to a global memory location.

When all validating threads are finished, the addresses that have been collected are posted to the website 'http://eaglehousing.com/images/1/out.php?a=upl'. The tool then sets the registry key to signal that the machine has already been searched.

Finally, the spam tool creates and launches a batch file, which removes the tool from the infected system.

CONCLUSION

The Bagle.BI compound followed a rather complicated infection cycle, which at first sight does not make much sense.

The worm downloads a list of target email addresses, then sends out a dropper which drops a downloader, which downloads files from predefined websites. Basically, the worm is being sent to web addresses that are in the control of the virus writers. In the first stage, most of these sites carried new Bagle variants and a network of infected hosts was built up, with the hosts waiting for an update to appear on any of the download sites.

In the second stage the sites carried email address collector tools, which provided the virus author(s) with email addresses gathered from the network they created.

So the virus authors have ended up with a large network of infected hosts and a large collection of verified email addresses – probably two of the most highly-prized possessions for virus writers nowadays. Long gone are the days, when virus writers' sole purpose was to see the names of their creations on *CNN*.

VIRUS ANALYSIS 2

IT'S ZELL(D)OME THE ONE YOU EXPECT

Peter Ferrie and Heather Shannon
Symantec Security Response, USA

It was a Tuesday and it was sunny outside, but I was inside waiting for my email client to finish retrieving messages. It was stuck on one mail that had a huge attachment: a sample of W32/Zellome.

W32/Zellome arrives as an email attachment. It seems to exist only to demonstrate its polymorphic engine, since the worm component is messy and platform-dependent.

EXTREME PROGRAMMING

The polymorphic engine takes an idea that was first used by W32/Apparition, but takes it much further. W32/Apparition carried its own Pascal source code, which it dropped on machines on which a Pascal compiler was found. Apparition would insert garbage instructions into that source code, before directing the compiler to compile it. Zellome, by contrast, carries the compiler as a component of its polymorphic engine. Additionally, Zellome's polymorphism is implemented in an unusual way: by using a genetic algorithm.

NATURAL SELECTION

Evolutionary algorithms, also known as genetic algorithms, are based on the idea of biological evolution. By combining characteristics from a predefined set (genes), and altering parts of them randomly (mutation), new offspring is produced with new characteristics. The less fit of these tend not to pass their genes on to succeeding generations. At least, that's the idea. There have already been viruses that have used this technique, including W32/Simile (see *VB*, May 2002, p.4).

Zellome uses a genetic algorithm for a different purpose. Traditionally in this analogy, the virus is treated as a species, replications of the virus represent individuals, and 'fitness' is the ability to survive in an environment populated by hostile anti-virus software. For Zellome, the genetic algorithm is not a model of virus replication; rather, it is just a computational technique used to produce a polymorphic decryptor. The species is a set of functions, and 'fitness' is how close the function comes to producing the required output.

Ultimately, we come to the question: why use a genetic algorithm in the first place? This is difficult to answer, because the results are, essentially, no different from the output of a standard polymorphic engine. It is really no

more difficult to detect than normal polymorphic code. It is highly obfuscated, but it has constant characteristics. It is effective against emulators, with its many iterations and heavy floating-point usage, yet its extremely ugly compiled code is a giveaway.

INCUBATOR

The virus author calls the polymorphic engine an 'incubator'. Whenever the incubator is run, it begins by randomly displaying a message box identifying the virus author's name and his choice of the virus name (however, the engine is simply a modified version of a free tool written by someone else). Next, it will check if it was run from the %windows% directory. If it was not run from there, it will copy itself to the %windows% directory as 'incubator.scr' and create 'incubator.txt' that contains the name of the original file. Then the incubator executes itself from the %windows% directory and then deletes the file named in the 'incubator.txt' file.

KEPT AFLOAT

Before generating a new worm, the incubator encrypts the worm code that is stored in its .data section. The basic encoding scheme substitutes an 8-bit value, x, with a 32-bit float value, E(x), where E is a random quadratic function. The .C, .D and .E variants of the worm also preprocess the data with another polymorphic engine (the same one used in W32/Zelly.B, created by the same virus author), before applying the substitution encoding.

With the worm encoded as an array of floats, the incubator now needs a decryption routine to decode the encoding. There are several ways to do this: one could use hash tables, construct an interpolating polynomial, or use some algebraic manipulation to solve the quadratic equation. Instead, however, Zellome does it the hard way: it uses a genetic algorithm to 'grow' a decryptor. It generates random arithmetic expressions, then mutates and combines them until it finds one that happens to undo the encryption function for the 256 possible input values. This is a time-consuming task that can take anywhere from five minutes to half a day.

Once it finds a decryptor, Zellome generates C source code containing the encoded worm and a short decryption loop, interspersed with about a megabyte of garbage code.

WHAT'S COOKING?

Zellome starts the incubation process by generating an initial population of 16,384 expressions. The basic elements of the expressions are:

- binary operators: *, +, -, /
- unary functions: exp, sin, sqrt
- constant float or integer values
- pi (just another constant)
- a variable, 'X', representing the input to the function

An expression is represented as a list of 256 tokens, thus the size of the generated decryptor is limited.

No filtering is done on these expressions: duplicates, synonymous expressions, and obviously unsuitable candidates such as constant functions, are all allowed.

RANDAMN

Due to an improperly seeded random number generator, the initial expressions are not actually as random as they should be. It is possible for the incubator to generate the same list of initial expressions in subsequent runs, depending on the value of an uninitialized variable that is passed to srand(). It still produces a different decryptor each time, because the encryption function is generated before this call to srand().

DARWINIAN EVOLUTION

The incubator then begins the process of creating new generations from this initial population.

First, the current generation is checked for a suitable decryptor function. It estimates the fitness of each expression, and saves the value for later use. The 'fitness' of a candidate is the sum of absolute distances from the target values, multiplied by -0.01. Expressions that produce any extraordinary values (such as infinity or 'not a number') are assigned a fitness of -FLT_MAX, effectively eliminating them from further consideration. Particularly promising expressions are checked against each possible input. If the resulting values all fall within 0.5 of the target output, a decryptor function has been found, so control passes to the source generation routine.

If a decryptor is not found, Zellome proceeds to select the next generation.

The population is kept constant at 16,384 individuals.

- 15 per cent are unmodified members of the current generation, including the three fittest specimens.
- 50 per cent are mutated individuals from the current generation.
- 35 per cent are new 'children', constructed by combining existing expressions.

When Zellome selects an expression for propagation, mutation, or breeding, it chooses four at random, then picks

the fittest of the four. Duplicates are allowed, so the same expression may be used more than once.

The following kinds of mutation can occur:

- Replace one constant with another
- Replace a subexpression with a new random expression
- Change the order of arguments (for example, change 1/X to X/1)
- Simplify constant expressions (for example, replace sqrt(4.0) with 2.0)
- Replace an operator or function with another (for example, change sqrt(X) to sin(X))
- Switch subexpressions (for example, change (X + 1.0) * 2.0 to (X + 2.0) * 1.0)

To keep the expression size from running over the 256-token limit, large expressions (those with more than 64 tokens) are not subject to mutation, except for constant-substitution.

To 'breed' two expressions together, Zellome selects a subexpression from one parent, and replaces it with a subexpression from the other parent. There is a five per cent chance that the offspring will be mutated.

For example:

1st parent: (sqrt ((X) / (5)))

Subexpression: (X) / (5)

2nd parent: ((sqrt (X)) / (2.649156))

Subexpression removed: ((sqrt (X)) / (...))

Offspring: ((sqrt (X)) / ((X) / (5)))

This process continues up to 10,000 generations. In practice, about 50 to 150 usually suffice to find a decryptor, though it can take much longer.

In testing, it was observed that all of the decryptors found by this method contained a sqrt() subexpression. This may be explained by the quadratic encryption function: intuitively, if you want to invert Ax^2 + Bx + C, it makes sense to start by taking the square root of the input. By contrast, only 25 per cent of the decryptors contained sin(): periodic functions are unlikely to be useful when inverting polynomials.

GETTING RESULTS

The incubator creates a file, 'result.c', in the current directory. It writes a constant preamble declaring some functions and global variables, emits the decryption code to a buffer (to be written to the file later), and writes a series of random functions that contain array assignments and garbage code.

The array assignments initialize a buffer with the encoded worm. (These values are treated as floats during the decryption computation, but they are initialized as an array of integers, probably to prevent rounding error.) Zellome does the assignments in random order. After it writes the decryptor function, the remaining array assignments access the buffer through a pointer to a random location in the middle of the array; the purpose of this obfuscation is not clear, but one possible explanation is that it is to make it more difficult to see that one assignment belongs to the same region of memory as another assignment.

The garbage code consists of function calls, arithmetic expressions, assignments to local variables, 'if' statements with random conditions, and 'for' loops that execute up to 1,000 times. A function call may optionally be enclosed in an 'if' or 'for' block, or both, but not if the function contains or calls any necessary code, such as an assignment function. This ensures that all of the non-garbage code is called exactly once.

Rather than assigning random names to functions and variables, Zellome observes the following naming convention:

l#### – local variable

p#### – parameter

d#### – array assignment or decryption function

f#### – other function

if#### – inline function

where the ####s are numbers assigned in increasing order.

(This systematic naming convention, together with the constant appearance of parts of the code, suggest that the author's design goals did not include concealing the source code from detection.)

MALFUNCTION

Functions take up to seven arguments with random types. They always return a value: there are no void functions. The return values are either saved to dummy variables, or discarded; they are not relevant to the decryption process.

Function calls can be any of the following:

- other random functions in the source file
- sqrt, exp, sin, abs, acos, asin, atan, atof, cos
- rand, srand
- fopen (but not fclose)
- malloc (allocating up to 65535 bytes – but not free)
- strcmp, strlen

- SetCurrentDirectoryA, CreateDirectoryA, CopyFileA, DeleteFileA, MoveFileA

The code ignores any errors returned by any of these functions. Since the parameters are well-formed, none of the functions would cause an exception to occur, so there is no need for critical error detection. However, the use of CreateDirectoryA() does create random directories, and the use of DeleteFileA() and MoveFileA() could, in exceptional circumstances, result in the deletion or renaming of real files.

Nestled in the middle of this random code, there is a surprisingly readable, un-obfuscated decryption loop which applies the decryption expression, saves the decoded worm to a stack buffer, and transfers control to the worm.

GET BUFF

Curiously, the code generator allocates a 10,000,000-byte buffer to contain this decryptor code, though the decryptor part itself is only a few hundred bytes long, and the entire source file is seldom longer than 1,500,000 bytes. It is possible that the goal was to generate more of the source 'genetically', but the design was scaled back to a smaller sub-problem: use a genetic algorithm to generate a numeric expression, but produce the rest of the code through normal means.

To give structure to the code, Zellome creates a series of call trees, each containing an arbitrary number of garbage functions, and one non-garbage function as a leaf node. It emits each tree to the file separately, first declaring each function that appears in the tree, then writing the functions, in breadth-first order, starting from the top of the tree.

Finally, it generates a top-level tree that calls all of the others – first the assignment parts, then the decryptor part. The root node for this tree is supposed to be WinMain, but due to a bug, this is not always the case. When it generates a new node for the call tree, it first decides whether the node will be a garbage function, and only later checks whether it should be WinMain. If it creates the non-garbage function as the root node of the last tree, the tree-generation routine thinks it has finished, so it exits without producing WinMain. In this case, the source will not compile.

DROP YOUR BUNDLE

The incubator drops the compiler files at this point, in order to compile the produced source code. The .A and .B variants are missing one key file, though (mspdb60.dll), so unless the file is present already on the system, all compilations will fail. When compiling, the incubator uses compiler switches chosen at random from a set that it carries.

The compiler switches that are used cover different areas. There are switches for code optimization (optimize for size, or for speed, or disable optimization entirely); for the expansion of inline functions (all, some, or none); for the emitting or omitting of frame pointers; for the presence or absence of exception handling; and the type of exception handling, if present.

Finally, the incubator will either compress the file with the copy of UPX that it carries, or append the incubator to the created file, but not both, then run the result. Since there is no detection of multiple instances, new replicants will continue to be generated for as long as the incubator is part of any replicant.

THE WORM HAS TURNED

The worm component begins by retrieving the list of APIs that it will use, some of which are not used, including two which are critical to prevent multiple copies of the worm running at the same time.

It copies itself to the %system% directory as 'bigfish.scr', then hooks the execution of Task Manager. This is done by creating the registry key 'HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\taskmgr.exe', and the value 'Debugger', whose data are set to point to the copied file.

This technique was first described by the virus writer GriYo as 'Execution redirection', and published in the eighth issue of the *29A* virus-writing magazine. The idea is that *Windows NT*-based systems run the process named in the 'Debugger' value, expecting it to control the application that is named in the key. The worm does not run the original file afterwards. This change continues to work in Safe Mode, so it is necessary to rename the file instead, in order to run it manually.

To improve the chance of being executed, the worm also attempts to create a value in the 'HKLM\Software\Microsoft\Windows\CurrentVersion\Run' key, which it names 'bigfish', and whose data also point to the copied file. However, the use of the seemingly incorrect API (RegSetValue() instead of RegSetValueEx()) causes *Windows* to create a subkey instead of a value. The result is that there is no execution via that method on any platform apart from *Windows 2000*. Perhaps this is the platform that the virus author uses, which is why he didn't notice the problem.

REGISTER NOW

After hooking the registry, the worm queries the registry value 'HKCU\Software\Microsoft\Internet Account

Manager\Default Mail Account', then uses that value to query the 'Accounts' key for the 'SMTP Server' value. This server will be used to send mail, if possible.

The email attachment can arrive in one of two forms. One form is simply the worm file, the other form includes the polymorphic engine as appended data. If the polymorphic engine is present, the worm will detach it into the current directory and execute it at this time.

The worm always uses the name 'incubator.scr' for the detached file. The detached file is an independent component and executes without reference to the worm file.

In any case, the worm will encode itself into base64 format – perhaps surprisingly, using the standard dictionary algorithm. It might be considered surprising because an alternative algorithm was published in the seventh issue of the *29A* magazine, which, at only 59 bytes in length, is smaller than the base64 dictionary itself. In fact, the worm code is taken from another virus by another author, with only a few modifications (single subject, etc.), but the same bugs.

The worm collects email addresses from two sources, and keeps a list of every email address that it finds. The worm does not avoid duplicated addresses. The first source of addresses is the file referenced by the registry key 'HKCU\Software\Microsoft\WAB\WAB4\Wab File Name'.

JUST BROWSING

The second source is the browser cache directory, within which all subdirectories will be searched for files whose extension is one of 'htm', 'asp', or 'xml'. For any such file found, if its size is between 512 bytes and 80 kilobytes, the worm searches within the file for the 'mailto:' string.

A number of bugs exist in this code – the most important of which is that, while parsing the file, the buffer pointer is updated to skip any address that was found, but the variable that holds the number of remaining bytes is not adjusted correspondingly. This can cause the routine to crash if at least one address exists, because the buffer pointer will fall off the end of the buffer. The crash is intercepted by the worm code, though, so the worm will continue to execute, but exit the address collection routine.

If the routine does not crash, potential addresses are examined for the presence of disallowed characters. If any such characters are found, then the worm will adjust the pointer in its collected address list to allow the next address to overwrite the invalid one. However, if no other addresses exist in the same file, then a bug causes the next address to be appended to the invalid address, instead of overwriting it.

HELO WORLD

After looking for email addresses, the worm attempts to resolve the address of the SMTP server. There is a critical bug here, which is the result of an incorrect assumption about the layout of certain networking structures. The worm assumes that the hostent structure, returned by the gethostbyname() API, is followed immediately by the address list. In fact, this is true for all *Windows* versions prior to *Windows XP*. In *Windows XP/2003*, there is a null pointer at that location. Thus, in all variants prior to .E, if run on *Windows XP/2003*, the code crashes at this point and never sends mail. However, on any earlier version of *Windows*, the code does work correctly. Additionally, the 'incubator.scr' file will still be running, if it was present.

In the event of a successful resolution, the worm will connect to the SMTP server. It was intended that the worm would check the return values from the server, however some of the branch instructions were removed, leaving compare instructions whose results are ignored. These compare instructions relate to the client initiation. The most likely reason for their removal is that the worm's domain string is malformed, and the worm author might not have worked out why a server would not return the expected response.

The worm chooses a random number prior to sending the message. This random number would be used to select between different sender addresses, subjects, message bodies and attachment names, however all of the conditions point to the same respective texts. This results in an email that always appears to come from 'Don Quijote y Sancho Panza', with subject 'juas juas cuidadin con el attachhhhrrrr!!!!!' (which translates roughly as 'heh heh watch out for the attachment!!!!!'), a message body of

```
juas juas juas peaso de bicho que lleva el attach!!!
juas juas!!! ;D

Vallez\29a
```

(which translates roughly as 'heh heh heh what a tiny bug is carrying the attachment!!! heh heh!!!'), and an attachment name of 'soyunpeasodebichooooooo.scr' (roughly 'I am a tiny buuuuuuug.scr'). The attachment will be the worm file. The worm will send a single email, but to multiple recipients. The recipients are all addresses found in the address book, and no more than 40 of the addresses found in files. After sending the email, the worm will exit.

CONCLUSION

As an unusual example of self-compiling malware and a novel misapplication of artificial intelligence, Zellome is an interesting specimen, but its many bugs and painfully slow execution time prevent it from working as a practical worm. In evolutionary terms, this species is heading for extinction.

TECHNICAL FEATURE

PROBLEMS IN STATIC BINARY ANALYSIS – PART I

Aleksander Czarnowski
AVET Information and Network Security, Poland

After reading Eric Uday Kumar *et al.*'s article 'DOC – answering the hidden 'call' of a virus' in last month's issue of *Virus Bulletin* (see *VB*, April 2005, p.7), I decided to share some of my thoughts on the problems related to performing analysis based on binary disassembly.

The background material presented in this article was collected during the development of our own executable binary audit solution. While such an audit is aimed at identifying dangerous code constructs and vulnerabilities, many of the techniques used are similar or even identical to those used in malware analysis.

INTRODUCTION

Static binary analysis techniques are becoming more advanced every year, and members of the security community are finding an increasing number of uses for these techniques. A good example is the use of binary analysis in the application security audit process.

In recent years we have also seen a great improvement in the tools supporting binary analysis – *IDA Pro* and *OllyDBG* spring to mind just to name some of the key players. Since these applications provide analysts with broad functionality – and in many cases an additional programming interface – the idea of using them as a foundation for more specific analysis tools seems a rational approach. After all, why reinvent the wheel?

Basing tools on *IDA Pro* is a very popular technique due to the fact that analysts have two options for extending its functionality: through IDC scripts or through plug-ins (using *IDA Pro SDK*).

IDC scripts are often used during analysis of encryption code: IDC is great for decrypting code that has been encrypted with simple encryption loops, like those based on the XOR algorithm.

For more advanced tasks plug-ins seem to be a more suitable option. Here, *IDA Pro* is used as a foundation and specialized plug-ins are written for particular tasks in binary analysis (as described in [1], for example). A set of tools for binary analysis based on *IDA* is also available for security analysts [2].

Finally, a third approach – based on analysing a binary object through analysis of its disassembly form – has been demonstrated by the authors of DOC [3]. The aim of this article is to describe some of the problems that may be encountered in developing tools based on static analysis and in the use of their output. For simplicity we assume limited user interaction during the generation of the disassembly listing.

SYSTEM LOADER VS DISASSEMBLER LOADER

Some readers might remember the discussion about the DOS EXE header signature when the INT 21h handler was disassembled. It turned out that EXE files could start with MZ or ZM bytes – prior to the full disassembly of the INT 21h handler there had been speculation that files with other signatures were treated differently. It soon became clear that the handler's author wasn't sure how bytes are stored in memory on x86 architecture so he did two checks for those bytes instead of one. This story demonstrates that even relatively simple system loaders can have bugs and specific behaviour that is not necessarily included in system documentation.

Today, executable objects are more complex, as are process loaders. This has some important implications for the analysis process.

Static analysis tools like disassembler must not only calculate the entry point properly, but also simulate some of the system loader actions. Many malware and copy protection schemes are based on different quirks in system loaders and the inability of disassemblers (read: *IDA Pro*) and debuggers (read: *OllyDBG* and *SoftICE* in the *Windows* environment) to load those files properly for analysis. This is the first problem we will analyse.

A good demonstration of the techniques mentioned above is in 0x90.exe, which was published as part of a challenge [4]. It is worth mentioning that submissions for this challenge also use dedicated *IDA* plug-ins to fight with obfuscated code.

I love this example as there are very few files around which are not malware and yet include so many different anti-analysis protection techniques. When an analyst is given a new binary object he usually tries to run it through his set of analysis tools. The 0x90.exe example is interesting because many popular tools (such as *IDA Pro*, *OllyDBG*,

Figure 1: 0x90.exe loading in IDA Pro prior to version 4.8.

dumpbin and *biew*) have problems inspecting the file. Prior to version 4.8, *IDA Pro* would hang for a few minutes before displaying the warning message shown in Figure 1. Figure 2 shows the result of loading the file into *IDA Pro 4.8* (the most current version of *IDA Pro* at the time of writing).

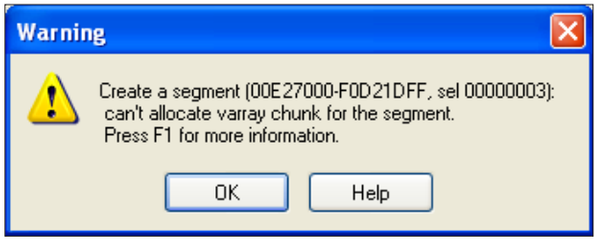


Figure 2: Warning message displayed on loading 0x90.exe in IDA Pro 4.8.

Here we come to the first limitation that has a direct influence on every process based on external disassembly: if we cannot load the binary into the analysis tool, we cannot perform any further actions on it.

On the subject of the *Windows* environment, it is also important to simulate the initial stack in order to perform a proper static analysis.

To demonstrate an example of using the stack layout provided by the system loader we will use the shortest assembly language program for the Win32 platform. The listing in Figure 3 is written in and can be compiled using the MASM32 package.

```
.386
.model flat, stdcall.

code
start:
     ret
end start
```

Figure 3: The smallest, fully functional Win32 application.

This sample demonstrates how processes can be terminated without calling the TerminateProcess() function. To understand how this 'RET' trick works we need to load the file into a debugger. Figure 4 shows the call stack for this application using *OllyDBG*. If the static analysis tool is not able to simulate the true execution environment provided by the system loader we might have some problems in achieving proper analysis results.

Coming back to our 0x90.exe example, it is important to analyse how it was protected against so many tools. Again, the secret lies in the difference between the real system process loader and those provided within tools like *IDA Pro* or *OllyDBG*. For further analysis we will use PE Editor from *PE Tools 1.5* [5]. Take a look at figures 5 and 6 – all

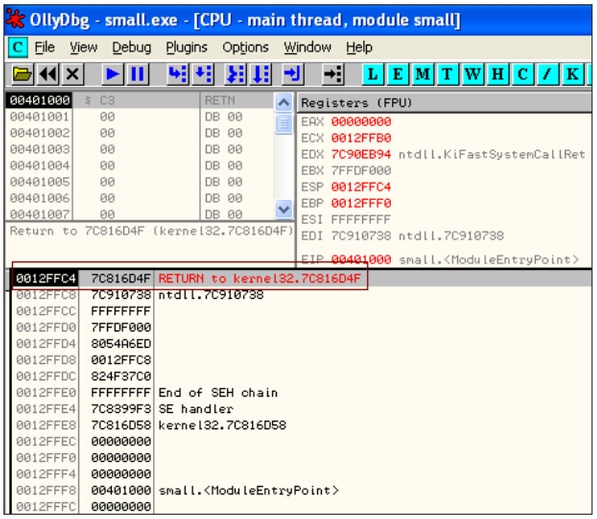


Figure 4: Call stack for Figure 3 code in OllyDBG.

bogus values for particular header fields are ringed. If you inspect Figure 6 closely you can also see that a non-standard ImageBase value is used – this field is not set to the

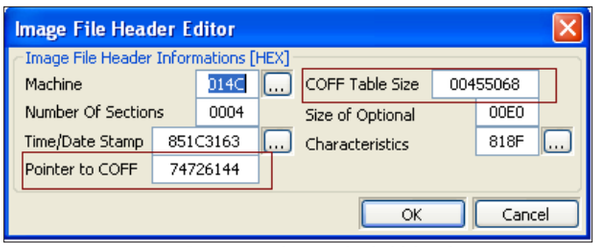


Figure 5: Bogus values in the 0x90 header.

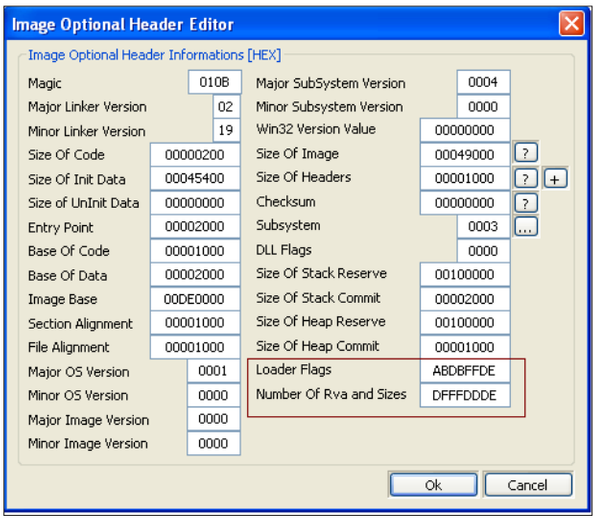


Figure 6: Using Loader Flags and Number of RVA sizes to protect the binary against debugger loaders and disassemblers.

standard 00400000 address (this will not make analysis with *IDA* or *OllyDBG* harder, however I suspect that some tools may always assume the default value without checking the real ImageBase value).

When analysing a binary object we need to understand its format fully. Another trick used in 0x90.exe to make its analysis harder with static methods can be seen by inspecting the section table.

As illustrated in Figure 7, the value of the Raw Size field for the NicolasB section is bogus and causes some disassemblers to try to allocate enormous buffers to store it in memory. This is why *IDA Pro* fails to load the file. Only after correcting the Raw Size value for this section is it possible to load the file into *IDA Pro* and *biew*.

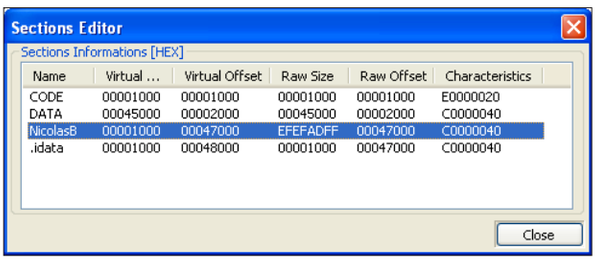


Figure 7: Inspection 0x90.exe section table.

HOW IS YOUR PROCESS LOADING TODAY?

When analysing program flow control it is important to follow it correctly. An obvious method for static analysis is to find the code entry point and go further from there. In the case of dynamic analysis the most trivial method is to place a breakpoint opcode (0xCC on IA32 architecture) on the first byte that is pointed to by the header field, as entry point and execute code.

Unfortunately both methods can fail due to the fact that the system loader can perform additional actions before passing control to the binary.

We can see the process of loading and running PE/EXE by the system loader using even a very simple debug loop like the one provided in [6]. If the binary is importing functions from external DLLs, the DLL initialization code will be executed before passing control flow to the binary entry point.

There is also another possibility due to the fact that *Windows* assumes that every application needs kernel32.dll. This results in kernel32.dll being loaded every time a process is to be executed. We can observe this behaviour if we load the example from Figure 3 under the control of a debugger. I used *OllyDBG*'s 'Executable modules' option (ALT+M) to produce the output shown in Figure 8. It is possible to modify kernel32.dll to execute additional code.

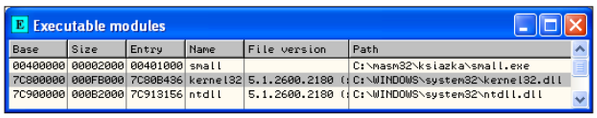


Figure 8: System loaded DLLs when import table is empty.

The way to avoid these problems is to analyse the import table of the binary first. By using table entries we can analyse the libraries that would be loaded and executed by the system loader. Note that the aforementioned problems have impacts on both dynamic and static analysis. If we miss a particular DLL in our disassembly analysis the results might be flawed.

IS YOUR DISASSEMBLY CORRECT?

In fact, flawed results of analysis based on disassembly are not usually caused by missing a particular library during the disassembly process, but rather due to obfuscated code or incorrect assumptions made by the disassembler.

It is important to note that some code constructions that we usually classify as obfuscation can be generated by a compiler. For example, we are used to calling functions using 'call' and returning using 'ret'/'ret n' instructions. Unfortunately compilers do not always generate code in this way.

```
  call   j__RTC_Initialize
  call   j__setargv
  cmp    __defaultmatherr, 0
  jnz    short loc_416B3E
  [...]
j__RTC_Initialize proc near ; CODE XREF: mainCRTStartup
  jmp    _RTC_Initialize     ;use jmp to call function
j__RTC_Initialize endp
  [...]
_RTC_Initialize proc near
  push   ebp
  mov    ebp, esp
  [...]
  mov    eax, [esi]
  test   eax, eax
  jz     short loc_417A08
  call   eax   ;use register for pointing destination
               ;address
  [...]
  mov    eax, 1
  retn
```

Figure 9: Calling functions with jmp and call eax.

Figure 9 shows code generated by *Microsoft Visual Studio.NET 2003 for C++* project with Debug settings. If we were to remove the comments stating that this code is called from mainCRTStartup, we could misidentify this code as some part of a simple anti-disassembly trick. This is due to the following:

- The procedure starts with an unconditional branch instruction (JMP) and there is no RET instruction at the end of this procedure.
- The procedure to which the JMP instruction points starts with the typical PUSH EBP/MOV EBP,ESP prologue – there is nothing special about this code with the exception of how we got here with help of the CALL/JMP trampoline. Searching code for PUSH EBP/MOV EBP,ESP is a very simple, yet effective, method of discovering procedures – even if there is no direct CALL instruction that points to the PUSH EBP address as in our example.
- The procedure is calling another procedure and dynamically calculating the address which is stored in the EAX register – in some cases without code emulation or single stepping/tracing we might not be able to calculate the correct address.

MORE PROBLEMS

Compilers are not the only source of problems. From time to time developers decide to change a convention. A good example is the hot-patching technology used by *Microsoft*. Breaking disassembly and stack analysis with code obfuscation techniques will be discussed next month in the second part of this article. In the second part we will also take a look at some emulation and tracing techniques used in binary object analysis.

BIBLIOGRAPHY

[1] Greg Hoglund and Garry McGraw, *Exploiting Software: how to break code,* Addison-Wesley 2004, ISBN 0-201-78695-8.

[2] *BinDiff*: http://www.sabre-security.com/products/bindiff.html.

[3] Eric Uday, Aditya Kapoor and Arun Lakhotia, 'DOC – answering the hidden 'call' of a virus', *Virus Bulletin*, April 2005, p.7.

[4] Scan of the Month 33: http://www.honeynet.org/scans/scan33/.

[5] *PE Tools 1.5*: http://www.uinc.ru/files/neox/PE_Tools.shtml.

[6] Writing the Debugger's Main Loop: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/debug/base/writing_the_debugger_s_main_loop.asp.

[7] *IDA Pro*: http://www.datarescue.com/.

[8] *OllyDBG*: http://home.t-online.de/home/Ollydbg/.

PRODUCT REVIEW

AHNLAB V3NET

Matt Ham

AhnLab's *V3Net* was last reviewed in *VB* more than two years ago (see *VB*, January 2003 p.18) – since when there have been considerable changes within the product.

AhnLab is based in South Korea. When I lived there eight years ago the Internet was just starting to become popular worldwide, but South Korea proved to be faster on the uptake than most. The government was keen, housing tended to be clustered together and there was an existing high technology manufacturing base – all of which were conducive to the development of high-speed Internet access. As a result, the Internet infrastructure in the country is among the best in the world – which is, of course, a double-edged sword considering the viruses, spam and other assorted unpleasantnesses that such a status brings with it.

It is not surprising, therefore, that anti-virus is not the only security offering from *AhnLab*. The company's flagship product is *V3Net*, which is available both for servers and desktops (both products were inspected in this review). However, the company also produces a wide range of products under the *Security ASP* banner, most of which are accessible through network connections rather than direct downloads or retail purchases.

The marketing material for *Security ASP* suggests that this offering may be aimed more towards ISPs and corporate users than for sale directly to end users. It offers web-accessible scanning for spyware, viruses and general security holes. There is a dedicated scanner for keyloggers, as well as heuristics and device driver hooks – this is the most dedicated, yet limited, malware scanner I have ever seen aside from those designed for a single virus.

Some of the product features are noted as having been designed for use in online games – this is the first time I have seen online gaming mentioned as an area in which security vendors are competing (but, given the popularity of such pastimes in South Korea, and the propensity for games players to seek advantages through any means, it is not at all surprising).

Although this range of additional security products is not a surprising area of expansion for a company such as *AhnLab*, I was surprised that these functions were not tied into *V3Net* and the whole sold as a combined suite. In fact, further investigation showed that the *AhnLab Security Pack for Desktops* does integrate personal firewall and anti-virus functionality, with 'additional features' which turn out to be data deletion and encryption tools. There is also a shareware *Palm* scanning product available for download.

WEB PRESENCE AND DOCUMENTATION

AhnLab's main website is at http://www.ahnlab.com/, though those unfamiliar with hangul would do well to use the pages located at http://info.ahnlab.com/english/. Chinese and Japanese versions of the site also exist, *AhnLab* being certified to supply mainland China.

The website offers the usual fodder of a corporate anti-virus site, but with a distinct skew away from the usual marketing product details. Unfortunately, there is a slightly irritating requirement to fill in personal details for each evaluation file required. This is the case even when documentation or sales materials are requested (the latter in particular seems to be a counterproductive move).

Electronic versions of the software were used exclusively during the review process, thus no hard copy documentation was available. The availability of help files was very mixed, the area in which they were most notably lacking was during the installation process and while using the Configuration Wizard or standalone Configuration application. These applications would benefit from at least a few hints as to what is affected by the choice of various options.

In contrast, the help function within *V3Net* itself is very good indeed. In the majority of those instances where I was left scratching my head as a result of confusing configuration choices, the help function was able to clarify matters. The help index has been created sensibly – offering a useful selection of the more frequently required subjects rather than the more common overload of keywords.

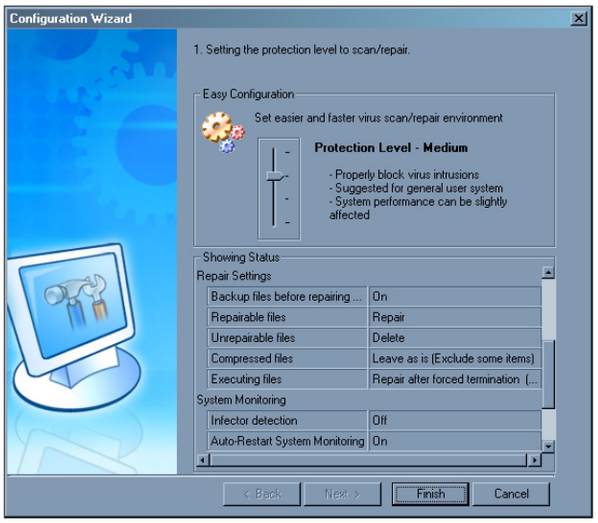
INSTALLATION AND UPDATE

Lab tests were performed using *V3Net for Windows Servers* on *Windows 2003 Advanced Server*. This package totalled 17.5 MB. The *XP* version, which was also tested for updating and interoperability with other anti-virus applications, was slightly larger at 18 MB.

The installation process is less involved than in many products, there being very few choices to make at any stage before the transfer of files has been completed. This allowed ample time for such delights as reading the licence agreement.

The contents of the licence agreement were much as expected, though there were a couple of rather more detailed caveats than those seen in the past. For one, the product is not guaranteed to work with new or updated operating systems – which, if Windows Update is used, could be any operating system at all.

The licence agreement goes even further by absolving responsibility specifically for damage caused by 'computer



virus, worm, spyware, Trojan horse, adware and scum discovered after the date of production of the software product.' Such denials of responsibility have always been common in licensing agreements industry-wide, though they seem to have become increasingly paranoid in recent years.

After the product has been licensed or specified as a trial version, the installation location is chosen and *V3Net* installs – the updater being added at a separate stage. At this point the user can choose whether to run the updater or perform a system scan.

This altogether standard part of the installation process is followed by the rather less usual Configuration Wizard (see figure above), which sets server parameters. This commences with the setting of a Protection level. The default level is Medium, with other options available by means of a slider bar. The effects of manipulating the bar are visible in more exact terms in the dialog box, though in some cases scrolling is required before the information becomes visible. The dialog in which this information is supplied is of fixed size, which does not help matters.

Default operation is with scanning enabled for all files, including archives, Internet accesses and startup programs. Some files are excluded, however, though it is not immediately apparent which ones. Disinfection will occur automatically, with the exception of archives. The on-access scanner will restart itself if terminated for any reason, and *V3Net* uses self-protection of some unexplained nature. Strangely, midday on Thursday is the default time for scheduled scans.

The three other setting available at this point are High, Low and Customized. The High setting removes all exceptions for scanning and implements a daily scheduled scan. The Low setting is very much more limited, scanning only local files and not offering automatic disinfection or a scheduled scan. If any of these preset configurations are chosen, the Configuration Wizard exits after confirmation.

The Customized option will be of much more interest to most administrators. This allows control over all the aforementioned fields independently. Many of the choices are self-explanatory and unsurprising, though others are much more interesting.

For example, when faced with a file which is infected and executing, the plan of action is less clear than simply detecting a file as infected and disinfecting it. An object which is in the process of execution may result in memory resident infection capability if it is left alone, but may cause users to gnash their teeth in frustration if it is shut down automatically to disinfect. Both of these actions are selectable, in addition to offering the user a choice as to whether to abort the program, or to force a reboot during repair in case memory is already infected.

A similar level of control is offered when dealing with compressed file formats. There is an option to scan all 27 that are supported by the engine, though each of these formats can be selected/deselected individually for scanning, should an administrator have reason to be so discriminating.

On-access scanning is also subdivided as to which objects are valid targets for detection. File accesses are, of course, the default. However, the bulk of the remaining choices seem to be subsets of this, since files selected from *Explorer*, downloaded files, *Microsoft Office* files and files registered through use of the Start menu all seem to be included already.

Scanning of specific areas when the screensaver is running does not fall so much into the category of on-access scanning so much as a very specific scheduled scan – but it may be activated here. Somewhat bizarrely, the page comes with a warning that functions to protect web browsers will not be effective if a web browser is not installed on the machine. Aside from being blindingly obvious, it would seem a challenge to find a *Windows* server that did not have browser functionality of any sort.

After these selections have been made, scheduled scans can be set up. For the purpose of the majority of these tests, however, the default setting of medium was used unless otherwise stated.

As stated previously, updating was tested on a *Windows XP* box using the *V3Net Pro* product. Installation here was essentially identical to that offered on the server platform. The update process may be performed from local folders or the *AhnLab* servers, the latter of which were used. Proxying is supported. In practice, the first update was very large,

taking some four or five minutes over an ADSL line. This is explained, at least in part, by the fact that the updates contained numerous engine modifications – a copious number of downloaded DLLs being a sure sign of this.

Downloads from the *AhnLab* sites over my UK-based connection were mostly quite sluggish, however, so the speed of download seems to be more a matter of bandwidth than the size of the update. Either way, the download times for subsequent signatures were almost unnoticeable.

FEATURES

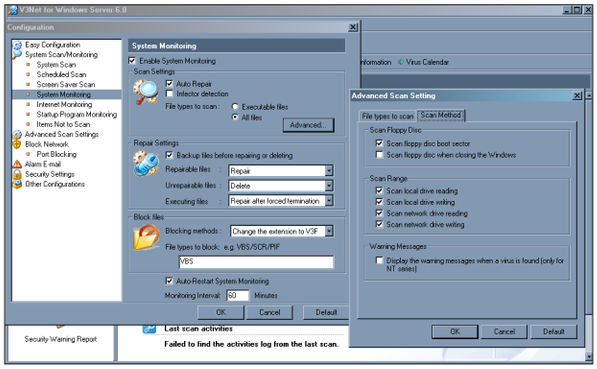
Once installed the presence of *V3Net* is obvious thanks to a desktop icon and a tray icon. The latter can hardly be missed, since it flashes yellow, cyan and green while active. Port blocking, general configuration, updates and the main program may all be invoked from here. Port blocking was not mentioned at any stage prior to its discovery here, so the feature came as something of a surprise. By default, port blocking is off, with no details supplied as to what happens when it is turned on.

The status of on-access scanning may also be set directly from the flashing icon. Deactivating scans for Internet downloads and startup programs does not affect this display, it is purely dependent upon whether file access scanning is activated. Deactivating this function leaves the icon flashing, but in monochrome. By default, however, the scan will be reactivated after 60 minutes.

Additions to the Start menu are also present, being divided between *V3Net* itself and the Smart Update Utility. The latter may be launched or removed from here. *V3Net* may also be launched or removed, and the Configuration function (which is not to be confused with the Configuration Wizard) and the previously unmentioned AhnReport are available too. With any luck AhnReport should remain unused by most users, since it is a utility for creating very detailed information about the location, updates and configuration of *V3Net* – its use is primarily for providing information to *AhnLab* should troubleshooting be required.

The Configuration application offers numerous tweaks to the product settings in addition to the Configuration Wizard invoked during installation. It seems odd that both applications exist, since the Configuration application would seem to easily replace most, if not all, of the functions of the Wizard.

A screenshot of the application in action (above) is fairly self-explanatory. The Easy Configuration option is simply a repeat of the slider-scale part of the Configuration Wizard. The Advanced Scan settings view, on the other hand, subdivides the general on-access scan into its components



which can be toggled independently. This area confirmed my suspicions that setting the Startup area scan is considered to be a part of the default on-access process, making its presence as a separate dialog more confusing. It is possible, therefore, to set the scanning of Startup items in two places with one setting having no influence upon the other.

Port blocking is also explained somewhat more by inspection of the control dialog. By default this is set to off, but since there are no designated ports to block, turning it on will have no effect. In order for ports to be blocked the user must create either a blacklist or a whitelist of ports manually. The help function in *V3Net* lists those ports which should not be blocked in order that vital functions are not interfered with, though these are not present automatically in the whitelist of the port blocker, which is an unhelpful omission.

The main *V3Net* application is where the bulk of the activity is based once configuration has been set. This is in the standard windowed view, with a left-hand pane for selecting one of six views, which then appears in the larger right-hand pane.

Functions are also available through drop down menus and icons towards the top of the interface, though the majority of these duplicate functions that are available elsewhere. Exceptions are the help function and links to various bits of information from the *AhnLab* website (specifically, the virus information, virus calendar and home pages of the site).

One notable and unusual addition to this upper area is Quick Scan. This allows a scan to be performed through the use of command line options, the default being 'C: /a /s', which will scan all files recursively on the C drive. It takes little inspiration to work out that this is using the commands as arguments to a command line scanner. Indeed, if '/?' is typed in as the argument, the list of command line options and the command line scanner name, V3Medic.exe, are revealed. The functionality is very limited, but this is a convenient interface for the fastest and most basic of scans.

The six views available are Home, System Scan, Event Log, Scan Log, Quarantine Station and Security Warning Report. Home supplies information on the status of the program, such as last update, results of the last scan and scheduled scans or updates. Schedules may be adjusted through links available here.

On-access scanning status in its various categories and port blocking are shown as on or off, though again these may be adjusted via links. In fact, all of these links lead to the Configuration application, mentioned previously as part of the installed Start menu programs.

The System Scan view, on the other hand, is a standard tree-based scanner. Customised scan lists may be set up for it, though the selecting of areas does not seem to offer files, only directories. This can be circumvented by using the Quick Scan option, though the scanning of files in long, convoluted paths is beset with problems using either method.

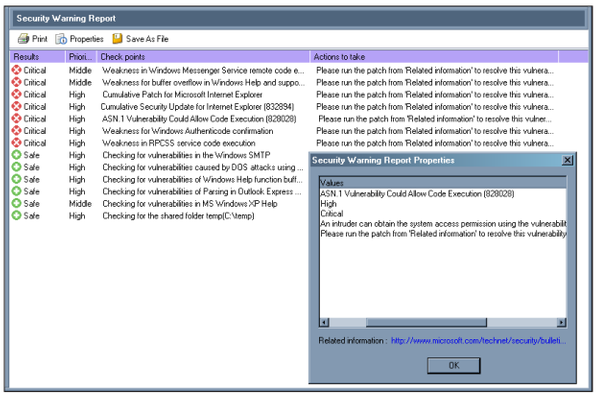
The view for the Event Log is one of those areas which remains dull until something woeful happens. During testing the only warning events noted were concerned with failure to connect to updates or the Internet in general on the isolated lab machines. The Scan Log likewise remains uninteresting unless viruses are present.

The descriptions given in the log file provide such information as the requirement for registry fixes in addition to file deletion or disinfection. However, there are no immediate instructions as to which registry setting must be altered. In most cases, of course, this will not have a major impact if the files are deleted, and lack of details can be seen as a space saving feature.

In cases such as W32/Blebla.B, however, the log also notes that the registry must be fixed or EXE files may no longer run correctly. Despite this warning, the files are removed if disinfection is selected. The situation here is rather complex, but the results of this disinfection could be catastrophic. This is a common problem with the scanners offered by many vendors.

The Quarantine Station view is very much self explanatory, leading to the Security Warning Report as the final part of the interface. This performs a basic security scan for issues which are commonly used by worms.

If a weakness is detected, information is provided including a link to the appropriate patch. The list of vulnerabilities is not large and lack of an automatic update mechanism could be considered a failing. With the *Windows* platform having numerous ways in which it can update itself automatically, however, *AhnLab* can easily be excused in this case. As might be expected, an unpatched lab machine triggered several alerts in this area, while a net-attached *Windows XP*



SP2 machine, set to automatically download from the Windows Update site, showed no known security issues.

CONCLUSIONS

One of the major impressions left by *V3Net* is that it is a complex product, offering a great deal of flexibility in the ways in which it is configured. With the server product selected for testing, the level of complexity was an expected side-effect of the requirements of an administrator. However there is much the same level of complexity in the *XP* version of the software, and an average user may well be confused by the options available. One might expect most such users to use the Configuration Wizard and not the other Configuration application, where matters are considerably simpler.

The lack of configuration help files is a little more of a concern, though where help is available it is very much improved over that experienced in the last review of *V3Net* – and comprehensive too. The translation team at *AhnLab* are also due some praise, since the odd translations in the product seen two years ago, have now vanished completely.

Overall, the product has matured well and it has performed increasingly well in *VB*'s Comparative tests. I just hope that *AhnLab*'s attention to security in online games will enable me to do some lengthy 'research' in that area.

Technical Details

Test environment: Identical 1.6 GHz Intel Pentium machines with 512 MB RAM, 20 GB dual hard disks, DVD/CD-ROM and 3.5-inch floppy drive running *Windows XP Professional* or *Windows 2003 Server Web Edition 5.2*. Athlon XP1600+ machine with 1 GB RAM, 80 MB hard disk, DVD/CD-ROM and ADSL internet connection running *Windows XP Professional Service Pack 2*.

Developer: *AhnLab*, 6th Fl, CCCM Bldg, 12 Yeouido-dong, Yeoungdeungpo-gu, Seoul 150-869, Korea; email customer@ahnlab.com; website http://info.ahnlab.com/english/.

END NOTES & NEWS

The sixth National Information Security Conference (NISC 6) will be held 18–20 May 2005 at the St Andrews Bay Golf Resort and Spa, Scotland. For more information see http://www.nisc.org.uk/.

AusCERT 2005 takes place 22–26 May 2005 in Gold Coast, Australia. Programme details and online registration are available at http://conference.auscert.org.au/.

The third International Workshop on Security in Information Systems, WOSIS-2005, will be held 24–25 May 2005 in Miami, USA. For full details see http://www.iceis.org/.

The 7th Protecting Critical Information Initiative (PCII) Convention takes place 6 June 2005 in London, UK. PCII is aimed at both public and private sectors and will cover a variety of information assurance-related topics including electronic and physical information threats, best practice, financial and legal reporting pressures, disaster recovery and contingency planning. For full details see http://www.pcii-initiative.co.uk/index.htm.

The 3rd annual BCS IT Security Conference takes place on 7 June 2005 in Birmingham, UK. The conference focuses on identity theft, hacking, cyber-terrorism, network forensics, secure web services, encryption and related topics. See http://www.bcsinfosec.com/.

NetSec 2005 will be held 13–15 June 2005 in Scottsdale AZ, USA. The programme covers a broad array of topics, including awareness, privacy, policies, wireless security, VPNs, remote access, Internet security and more. See http://www.gocsi.com/events/netsec.jhtml.

A SRUTI 2005 workshop entitled 'Steps to Reducing Unwanted Traffic on the Internet' takes place 7–8 July 2005 in Cambridge, MA, USA. The Usenix-sponsored workshop aims to bring academic and industrial research communities together with those who face the problems at the operational level. For more information see http://www.research.att.com/~bala/sruti/.

Black Hat USA takes place 23–28 July 2005 in Las Vegas, NV, USA. The deadline for submitting paper proposals is 1 May 2005; registration for the event is now open. For details see http://www.blackhat.com/.

The 14th USENIX Security Symposium will be held 1–5 August 2005 in Baltimore, MD, USA. For more information see http://www.usenix.org/.

The Network Security Conference takes place 19–21 September 2005 in Las Vegas, NV, USA. The conference is designed to meet the education and training needs of the seasoned IS professional as well as the newcomer. For details see http://www.isaca.org/.

The 15th Virus Bulletin International Conference, VB2005, will take place 5–7 October 2005 in Dublin, Ireland. The programme for the three-day conference can be found on the *VB* website. For more information or to register online see http://www.virusbtn.com/.

RSA Europe 2005 will be held 17–19 October 2005 in Vienna, Austria. For more details see http://www.rsaconference.com/.

WORM 2005 (the 3rd Workshop on Rapid Malcode) will take place 11 November 2005 in Fairfax, VA, USA. The workshop will provide a forum to bring together ideas, understanding and experiences bearing on the worm problem from a wide range of communities, including academia, industry and the government. The organisers are currently seeking submissions from those wishing to present at the workshop. Full details can be found at http://www1.cs.columbia.edu/~angelos/worm05/.

The eighth Association of Anti-Virus Asia Researchers International Conference (AVAR 2005), takes place in Tianjin, China on 17 and 18 November 2005. The theme of this year's conference will be 'Wired to Wireless, Hacker to Cybercriminal'. The organizers are currently seeking submissions from those wishing to present at the conference, the deadline for submissions is 10 June 2005. For more details see http://aavar.org/

Infosecurity USA will be held 6–8 December 2005 in New York, NY, USA. The conference will take place 6–8 December, with the accompanying exhibition running 7–8 December. For more details see http://www.infosecurityevent.com/.

SUBSCRIPTION RATES

Subscription price for 1 year (12 issues) including first-class/airmail delivery: £195 (US$358)

Editorial enquiries, subscription enquiries, orders and payments:

Virus Bulletin Ltd, The Pentagon, Abingdon Science Park, Abingdon, Oxfordshire OX14 3YP, England

Tel: +44 (0)1235 555139 Fax: +44 (0)1235 531889
Email: editorial@virusbtn.com Web: http://www.virusbtn.com/

vb Spam supplement

CONTENTS

NEWS & EVENTS

UN DISCUSSES SPAM

The United Nations (UN) has revealed that discussion at the third meeting of its Working Group on Internet Governance focused on spam, network security and cybercrime. The consensus among participants was that, although the topic of spam is not yet officially on the international agenda, it must be discussed as a matter of priority. The question of how to deal with spam and protect the Internet was discussed, with emphasis on the need for a multifaceted and multi-layered approach. A number of proposals were put forward, ranging from drafting model legislation to more informal models of collaboration. The Working Group will put together a report 'for consideration and appropriate action' which will be submitted to the UN Secretary General in July 2005 ahead of the second phase of the World Summit on the Information Society (WSIS), which takes place in November.

EVENTS

INBOX IT takes place 1–2 June 2005 in San Jose, CA, USA. The event will focus on all aspects of email including spam, phishing, zombies, outbound controls, encryption and the latest in new security technologies and techniques. More information is available at http://www.inboxevent.com/.

CEAS 2005, the Second Conference on Email and Anti-Spam, will be held 21–22 July 2005 at Stanford University, CA, USA. For more details see http://www.ceas.cc/.

TREC 2005, the Text Retrieval Conference, will be held 15–18 November 2005 at NIST in Gaithersburg, MD, USA. The conference includes a new track on spam. For more details see the following article and http://trec.nist.gov/.

FEATURE

CHALLENGES IN SPAM FILTER EVALUATION

Gordon V. Cormack
University of Waterloo, Canada

[This year, for the first time, the Text Retrieval Conference (TREC) will include a track on spam. The organizers of the spam track are looking to establish measures that will increase the availability of appropriate spam filter evaluation techniques. Here, track coordinator Gordon Cormack describes the challenges in spam filter evaluation.]

Do spam filters work? Which is the best one? How might filters be improved? Without standards, one must depend for the answers to these questions on unreliable evidence such as subjective impressions, testimonials, incomparable and unrepeatable measurements, and vendor claims.

You might think that your spam filter works well and couldn't be improved. Are you sure? You may think that the risk of losing important mail outweighs the benefit of using a filter. Could you convince someone who holds the other opinion? If I told you that my filter was 99 per cent accurate, would you believe me? Would you know what I meant? Would you be able to translate that 99 per cent into the risk of losing an important message?

The 2005 TREC Spam Evaluation Track will address some of these questions. Filters will be evaluated, but our purpose is not to crown the 'King of Filters'; rather it is to establish measures and evaluation techniques that are appropriate for ongoing filter evaluation, comparison, and improvement. To this end, participants will submit filters to be tested in a controlled environment, and evaluated using standard measurements. Through this effort we will begin to discover what works and what doesn't – not only in terms of filtering techniques, but also in terms of measurement techniques.

CHALLENGES IN FILTER TESTING

In order to evaluate a filter, it is necessary to subject it to some sort of test and to measure the outcome. The design of appropriate tests and measurements presents several challenges. In particular, a good filter test should meet three criteria:

- The test should model real filter usage as closely as possible.
- The test should evaluate the filter using measurements that reflect its effectiveness for its intended purpose.
- The test should avoid uncontrolled differences and yield statistically valid results.

REAL FILTER USAGE

A user's incoming email messages are received by the filter, which places them into one of two files: the *ham* file or the *spam* file. The recipient reads the ham file regularly, rejects any spam messages (which have been misfiled by the filter), and reads or otherwise deals with the remaining ham messages. The user may also report the misfiled spam to the filter. Occasionally they may search the spam file for ham messages that have been misfiled and report these misfilings to the filter. The filter may use this feedback, as well as external resources such as blacklists, to improve its effectiveness.

A filter test must capture or simulate the components of this setup:

- Incoming mail used in a test situation should emulate the mail that really is delivered – that is, it should be chronological, contain ham and spam addressed to the same recipient, contain its original header information, and should not be reformatted or contain any annotations added by the mail client or the user.
- In a real-world situation the user may report some or all ham and spam misclassifications to the filter. Some delay may be involved. The filter may not solicit or accept user feedback. Some of the user's feedback may be wrong. Users may be involved in more complicated feedback protocols. They may maintain whitelists and blacklists, or tweak any number of filter parameters in a (generally uncontrolled) effort to improve its performance. The filter may separate the messages into three, or ten files instead of two, with different user behaviour for each. A filter test must take these variables into consideration.
- The filter itself must be amenable to testing. Many filters lack interfaces that make it easy to identify or simulate the incoming email stream, placement of messages in folders, user feedback and configuration, and interaction with external resources.

INTENDED PURPOSE

A perfect filter would place all ham in the ham file and all spam in the spam file. In reality, however, some fraction of the spam will be delivered to the ham file (this is the spam misclassification fraction, or *smf*), and some fraction of the ham will be delivered to the spam file (the ham misclassification fraction, or *hmf*). The effectiveness of a filter may be characterized (in part) by these two fractions – in both cases the smaller the better.

Spam misclassification is an annoyance. After all, the purpose of the filter is to block this stuff, so smf indicates the extent to which the filter is not doing its job. Ham misclassification is more than an annoyance. It introduces a risk that you will lose an important message, so hmf also indicates the extent to which the filter is not doing its job.

A state-of-the-art filter might exhibit smf = 1.5 per cent and hmf = 0.05 per cent. That is, it would eliminate 98.5 per cent of the spam at the expense of eliminating one in every 2,000 ham messages. The impact on the user of eliminating that one message in 2,000 depends on:

- How likely it is that the recipient will retrieve it from the spam file, or receive the same message by a different channel.
- The consequences of losing the message – the impact of a missed advertisement from one's travel agent is likely to be considered less serious than that of a missed message informing the recipient about a flight cancellation.

ACCURACY

Magazine testers are fond of single-number scores that purport to characterize the overall 'goodness' of whatever is being tested. For spam filtering in particular, it is inappropriate, but all too common, for testers and their reports to conflate these two measures of effectiveness into one: *accuracy*.

Accuracy is defined as the fraction of all messages (ham or spam) that are *not* misclassified. Unfortunately, this measure is, essentially, useless for filter evaluation. Consider our hypothetical filter (where smf = 1.5 per cent, hmf = 0.05 per cent) – what is its accuracy? Or consider a filter claimed to be 99 per cent accurate – what are its smf and hmf? The answer, in each of these cases, is: without further information we do not know.

In order to deduce accuracy from hmf and smf we must know the proportion of ham and spam delivered to the filter. If the messages received by our user are 80 per cent ham and 20 per cent spam, the accuracy is:

(1 - [0.8*hmf] - [0.2*smf]) = 99.66 per cent

If the user receives 20 per cent ham and 80 per cent spam, the accuracy drops:

(1 - [0.2*hmf] - [0.8*smf]) = 98.79 per cent

Has the filter's performance changed? Of course not. The only difference is the ratio of incoming ham/spam.

Deducing hmf and smf from accuracy is even more dicey. With 80 per cent incoming spam, '99 per cent accuracy' might mean hmf = 0.01 per cent and smf = 1.3 per cent, which is pretty good, or it might mean hmf = 3.0 per cent and smf = 0.5%, which is very poor (no user would tolerate losing one in 33 legitimate messages).

OTHER MEASUREMENTS

The effectiveness of a filter may be captured as four counts, shown in the contingency table below, where *a* denotes the number of correctly classified ham messages, *b* is the number of incorrectly classified spam messages, *c* is the number of incorrectly classified ham messages, and *d* is the number of correctly classified spam messages. High values of *a* and *d* are good; high values of *b* and *c* are bad:

	True Classification	
Filter Classification	Ham	Spam
Ham	*a*	*b*
Spam	*c*	*d*

There is a remarkably large number of ways in which these four values can be combined, but only a few make sense as measures of filter effectiveness. The three measures discussed above are defined as:

hmf = c/(a+c)

smf = b/(b+d)

accuracy = (a+d)/(a+b+c+d)

Other measures, such as *(spam) precision = d/(c+d)*, or *weighted accuracy* = $(w_1a + w_2d)/(a+b+c+d)$ suffer the same defect as accuracy – they measure the incoming ham/spam ratio as much as the performance of the filter.

Many vendors report, by a variety of names, the fractions *b/(a+b+c+d)* or *c/(a+b+c+d)* – that is, spam or ham misclassifications as the fraction of all messages received. However, these values under-report the true misclassification fractions and, like accuracy, conflate effectiveness with ham/spam ratio. I will not justify these fractions with names, other than to note that they are *not* false positive and false negative rates, as often claimed. The terms *false positive* and *false negative* come from established methods used in medical testing, which are based on signal detection theory. If we think of spam as diseased email and of ham as healthy, false positive rate is a synonym for hmf; false negative rate is a synonym for smf. Any other use of these terms is incorrect.

The performance of a filter may be evaluated using curves that indicate the filter's performance under varying conditions. The TREC organizers intend to compute learning curves, which show the change in hmf and smf as messages are processed and user feedback is accumulated. Receive Operating Characteristic (ROC) curves will also be computed; these will plot the trade-off between hmf and smf for filters with adjustable sensitivity.

WHAT IS SPAM?

It seems odd that we have progressed this far without defining spam. In order to achieve precise measurement, we must have a precise definition. Given a definition, we need a way of adjudicating whether or not each message meets the definition. We call the result of this adjudication a 'Gold Standard' – it is not quite the 'True Classification' required for the contingency table, but it is as close to it as we can reasonably get. The quality of evaluation depends on three factors: setting a precise definition that realistically approximates the operational one, adjudicating messages with respect to this definition, and assessing the magnitude and impact on evaluation of adjudication errors.

TREC's tentative definition for spam is: 'unsolicited, unwanted email that was sent indiscriminately, directly or indirectly, by a sender having no current relationship with the recipient'. Under this definition, viruses are considered spam, as are 'unable to deliver' messages resulting from spam sent with a forged sender address. Political or religious bulk mail would be considered spam under this definition, provided the sender had no relationship with the recipient, and the recipient didn't want it. While we believe that this definition is reasonable, the important thing for testing purposes is that the definition is fixed and amenable to adjudication.

Given this (or any) definition of spam, a gold standard must be constructed for the incoming messages in the test. If the messages are archived, a person may examine them later and render an opinion as to whether or not each message meets the definition. As in a court of law, this opinion should be as to whether or not the message meets the definition, not as to whether or not the definition is appropriate.

Our definition of spam includes the relationship between sender and recipient, and the welcomeness of the message to the recipient. Therefore, one recipient's ham may be another's spam. This is not to say that spam is whatever the recipient thinks it is. The definition remains invariant, but the recipient may be in the best position to assess these two aspects of the definition.

THE GOLD STANDARD

It is impractical to read tens of thousands of email messages for the purpose of adjudication. The process would be

time-consuming, tedious and error-prone and would likely yield a higher misclassification rate than that of the filters being tested.

An alternative, that is used all too frequently, is to assume that the user's feedback is correct – that every ham and spam misclassification is reported. If this assumption were true, it would be a simple matter to construct a gold standard. One would simply capture the filter's judgements and correct them according to the user's feedback.

This approach can result in egregious evaluation errors. First, both hmf and smf will tend to be under-reported – either because the user failed to notice, or because they noticed but failed to provide the appropriate feedback. It is not unreasonable to assume that one per cent of all spam misclassifications go unreported.

This under-reporting is of little consequence in evaluating the current filter, but as a gold standard the error is disastrous. Suppose the current filter had hmf = 0.05 per cent and smf = 1.5 per cent, that the ham/spam ratio was 20/80, and that the user failed to report one per cent of all spam misclassifications. In the gold standard, 0.06 per cent of all messages adjudicated as ham would, in fact, be spam. A filter comparable to the current one, with hmf = 0.05 per cent, would have a reported hmf = 0.11 per cent – more than double its true error rate.

An iterative process can lead to a vanishingly low rate of undetected misclassifications in the gold standard. One of the techniques above is used to create a preliminary gold standard, G_0. One or more different filters are run on the same messages, using G_0 to simulate user feedback. Cases of disagreement – where the filter classifies the message as spam but G_0 says the message is ham, or vice versa, are captured. These cases of disagreement are adjudicated and, where the adjudication disagrees with G_0 they are corrected to form G_1. This process may be repeated to form G_2 and so on to G_n. The only messages that could potentially remain unadjudicated are those agreed upon by all filters.

For TREC, we use G_{n-1} to simulate the feedback of an ideal user and G_n as the gold standard for evaluation. Ideally, $G_{n-1} = G_n$ but a small amount of disagreement is acceptable as it affects only the simulated feedback, not the calculation of hmf and smf.

SCIENTIFIC CONTROL AND STATISTICAL VALIDITY

The sales pitch for one spam filter states:

> 'While most commercial solutions only claim a mere 95% accuracy (1 error in 20), a majority of [our] users frequently see around 99.95% (1 error in 2000) and can sometimes reach peaks as high as 99.991% (2 errors in 22,786, as with one particular user).'

Unfortunately, it is meaningless to compare accuracy, or even a more useful measurement, unless the filters are compared in similar circumstances; that is, with comparable test configuration, email to be filtered, user responses, spam definition, and adjudication. One way to make these factors comparable is to make them identical. Another way is to choose the email, users, etc. at random from some common source population. In both situations, we declare as significant only those differences that are unlikely to be due to chance. Statisticians use p to denote the estimated probability that an observed difference is due to chance, and the threshold $p < 0.05$ is commonly used to declare a result significant.

However, it is easy to lie with statistics. I just flipped a coin and it came up heads five times in a row. The probability of that happening is 1/32, so I've proved that the coin is unfair ($p < 0.03$) right? Wrong. Because what I really did was to flip a coin repeatedly until it came up the same five times in a row. But I only had to flip seven times in total until the last five were heads. Remarkable? No. There are 128 ways that seven coins can land and 16 of them involve five-in-a-row. So my seven-coin result yields $p < 0.13$ – slightly lucky but no evidence against the coin's fairness. The next time I tried it took me 17 attempts. Statistics are not useful unless you predict the result in advance and the results of all tests are considered, not just the 'significant' ones, or the ones that agree with your prediction. As a matter of policy, TREC reports the results for all systems in all tests; the testimonial above does not.

SETTING THE STANDARDS

Current spam filter evaluation techniques are riddled with inaccuracies and pitfalls. The 2005 TREC Spam Evaluation Track aims to provide a standard evaluation of current and proposed spam filtering approaches, to establish an architecture and common tools and methodology for an open-ended network of evaluation corpora (public and private), and to lay the foundation for more general email filtering and retrieval tasks. As a step towards achieving these goals, TREC will test spam filters in a controlled environment using a framework with standard components and measures. Through this effort we hope to begin to discover what works and what doesn't – both in filtering techniques, and in measurement techniques.

[TREC 2005 takes place 15–18 November 2005 at NIST in Gaithersburg, MD, USA. Participant guidelines for the Spam track at TREC 2005 and the Spam Filter Evaluation Kit are available at http://plg.uwaterloo.ca/~gvcormac/spam/.]