JULY 2004

virus BULLETIN

The International Publication on Computer Virus Prevention, Recognition and Removal

CONTENTS

ISSN 0956-9979

IN THIS ISSUE

BITING AT THE ANKLES OF IA64

Rugrat uses a fairly simple idea: take 32-bit code and port it to 64 bits – but the devil is in the details. Péter Ször and Peter Ferrie have the finer details of W64/Rugrat, the first known virus for the 64-bit *Windows* operating system on the *IA64* platform.

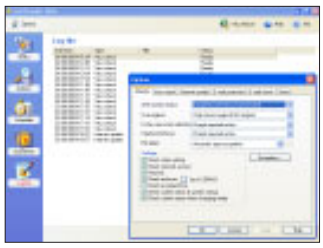
page 4

FEELING PATRIOTIC

What happens when you get a highly localised malware attack but most of the users in the country rely on software produced by the AV vendors for whom such an outbreak is barely noticeable? Gabor Szappanos urges the global AV players not to underestimate localised threats.

page 6

AVK REVIEWED



Matt Ham finds that *AVK* has been developed and documented clearly and concisely, making it a product that would be easy to use for a relative novice.

page 16

vbSpam supplement

This month: anti-spam news & events; Terry Sullivan reveals it's a small (spam) world; ASRG summary.

"The volume of Win32 threats seems to be growing faster than ever."

Péter Ször
Symantec Security Response

EPOCalypse NOW!

The general public may be wondering whether *Microsoft*'s repeated rewards for the heads of virus writers have had any effect on the global virus-writing landscape in 2004. Others are probably waiting to experience sweet revenge.

Given that friends will not always act like friends – especially when it comes to the temptation of a reward in exchange for information – some might expect virus writing to have been discouraged by the existence of such a bounty. On the other hand, conspiracy theorists might imagine that virus-writing groups encourage worm-writing contests just to collect the rewards – or even attempt to raise the rewards in order to demonstrate that they are the real bad guys, just like in the old Wild West.

So what has happened to the virus-writing landscape in 2004? Well, the number of Win32 virus threats this year grew by a whopping 300% over the same period last year, resulting in almost 4,500 variants this year so far. That said, there were a total of 5,500 Win32 creations in 2003. I am not going to get into the 'good old days' talk, but in 2001 (the year of CodeRed and Nimda) there were 741 new Win32 variants during the entire year. In contrast, there were over 700 new Win32 variants during just the first two weeks of June 2004. Thus, the volume of Win32 threats seems to be growing faster than ever, and virus writers do not appear to be afraid of creating more. The question is: would the situation be even worse without *Microsoft*'s reward announcements?

Where does the quick growth come from, you might ask. The answer is in the Gaobot, Randex and Spybot families that have all reached beyond the triple 'A' variations – there are over 1,200 variants in each family.

Recently, the media reported that 'the author' of Gaobot, as well as someone 'associated' with Randex, had been arrested as a result of successful police raids. However, even weeks after these reports the number of new variants belonging to these families showed no decline. In fact, Gaobot is widely distributed in source form, resulting in a situation that is worse than the effects of the popular use of virus construction kits. Indeed, Gaobot variants have more than a dozen exploits, stealth and some primitive polymorphism as well, not to mention that they are packed sometimes even five times if not more. Clearly, Gaobot variations have been developed by a number of people. In addition, the distributed source code suggested that newer editions of Gaobot were offered for sale via *PayPal* payments.

In fact, Gaobot variants introduced the exploitation of the LSASS vulnerability before Sasser appeared. Once Sasser came out using the same exploitation, Gaobot introduced a vampire attack against Sasser by hijacking its propagation routine, forcing it to propagate the Gaobot code. Next, Dabber variants exploited the vulnerability in the 'FTP server' code of Sasser. The worm war is far from over.

So what is the situation in the virus labs? Some of us think it is too difficult to get through a paradigm shift to handle Win32 from now on. Others attempt to measure up to the challenge, but the expectations are so high that they quickly decide to get involved in something different instead.

And what if there were MSIL EPO and metamorphics, 64-bit *Windows* viruses on IA64 or even mobile phone worms that spread via Bluetooth? Where would you hide? And what if this were all for real?

Working in the anti-virus industry has always required dedication, and this is what we all need even more now. Be dedicated to prevent the EPOCalypse!

[See p.4 for Péter Ször and Peter Ferrie's analysis of W64/Rugrat, the first known virus for the 64-bit Windows operating system on the IA64 platform. Next month's Virus Bulletin will contain the Peters' analysis of Cabir, the first virus capable of spreading via mobile phone - Ed]

NEWS

OBITUARY: MAREK SELL

On 12 June 2004 Marek Sell, creator of the Polish *MkS_Vir* anti-virus, died.

I met Marek somewhere around 1990, two years after he released the first public version of *MkS_Vir*. There are very few people in the AV industry who have stayed around for so long. If you remember the days of PC XT, debug, Periscope and Quaid Analyzer you'll know exactly how ancient this history is. *MkS_Vir* survived the switch from communism to capitalism, and *MkS* as a company evolved without any venture capital or outside investments – again, something that is unusual in the AV industry. All of this happened as a result of Marek's vision, deep knowledge and unquestionable ethics.

But that was only the business-side of Marek and his company. On the personal side, he was a great friend, always looking for new challenges and always ready to help anyone who asked. I loved chatting with him about assembly language issues and internal designs of *MkS* – in fact, you could speak with Marek about any subject, and he would always know some fascinating facts. He was one of very few people I know of who, in the days of *Windows XP*, was still writing tools for his work in assembly language.

Since 1997, Marek and I met every year at the *Virus Bulletin* Conference. Marek Sell attended all the *VB* conferences from the very first one. Only a few weeks ago – when Marek was going for a final treatment at the hospital – we made arrangements to travel to Chicago for VB2004. Now Marek is on a very different journey. His death is a huge loss to the whole AV community and especially to the Polish IT security community. Personally, I have lost great friend, but I am happy that I had a chance to meet Marek and be his friend. Marek, we will miss you.

Aleksander Czarnowski

AV GOING MOBILE

Following the appearance of SymbOS/Cabir.A, the first virus capable of spreading via mobile phone, mobile providers have been clamouring to become the first to offer anti-virus protection for mobile phones. *SK Telecom* announced the development, in cooperation with *AhnLab*, of *V3Mobile* anti-virus software, while *TSG Pacific* claimed to have its anti-virus solution ready and waiting – a spokesman said: "We anticipated this type of virus nearly two years ago." Meanwhile, mobile security company *Jamanda* has made a fix for Cabir available free of charge on its website. All this despite the fact that, to date, there are no confirmed reports of Cabir in the Wild.

Prevalence Table – May 2004

Virus	Type	Incidents	Reports
Win32/Netsky	File	248,490	83.63%
Win32/Bagle	File	34,677	11.67%
Win32/Sober	File	8,657	2.91%
Win32/Dumaru	File	2,109	0.71%
Win32/Klez	File	658	0.22%
Win32/Lovgate	File	269	0.09%
Win32/Mimail	File	184	0.06%
Win32/Funlove	File	178	0.06%
Win32/Fizzer	File	150	0.05%
Win32/Swen	File	136	0.05%
Redlof	Script	134	0.05%
Win32/Bugbear	File	114	0.04%
Win32/Valla	File	110	0.04%
Psyme	Script	96	0.03%
Win32/Sobig	File	86	0.03%
Win32/Parite	File	82	0.03%
Win32/Hybris	File	77	0.03%
Win32/Yaha	File	77	0.03%
Win95/Spaces	File	70	0.02%
Win32/Mydoom	File	66	0.02%
Win32/MyWife	File	65	0.02%
Win32/Nachi	File	63	0.02%
Win32/Magistr	File	58	0.02%
Fortnight	Script	49	0.02%
Win32/Mylife	File	42	0.01%
Win32/BadTrans	File	37	0.01%
Win32/Elkern	File	35	0.01%
Win32/Sasser	File	29	0.01%
Win32/Gibe	File	27	0.01%
Win32/Nimda	File	27	0.01%
WYX	Boot	23	0.01%
Laroux	Macro	21	0.01%
Others[1]		47	0.07%
Total		297,115	100%

[1]The Prevalence Table includes a total of 240 reports across 47 further viruses. Readers are reminded that a complete listing is posted at http://www.virusbtn.com/Prevalence/.

VIRUS ANALYSIS 1

64-BIT RUGRATS

Peter Ferrie and Péter Ször
Symantec Security Response, USA

On 26 May 2004, we received the first known virus for the 64-bit *Windows* operating system on the *Intel Itanium* platform. We decided to call it W64/Rugrat.3344.A.

Just like some of its predecessors (specifically W32/Chiton – see *VB*, June 2002, p.4), Rugrat is aware of Thread Local Storage, helping it to make the first successful tiptoe towards painless infection of *Windows* DLLs – at least in the .B variant of the virus.

BLAST FROM THE PAST

As might be expected, the text in the virus body suggests that Rugrat and Chiton share the same author: 'Shrug - roy g biv', who is now a member of the notorious 29A virus-writing group.

W64/Rugrat uses a fairly simple idea: take the 32-bit code and port it to 64 bits – but the devil is in the details.

Writing assembly for the *Itanium* is not simply a case of an everyday port of a 32-bit C application to 64-bit, which even your grandmother could do with a little advice. This is in contrast to the impression we were given when *Microsoft* introduced the platform with a demonstration: "Here is Larry who ported a million lines of C code in two weeks!".

Obviously Larry was not the kind who used to cast his pointers with DWORDs in front. Larry probably did not need to port a GUI either, and he obviously was not interested in writing a memory scanner to scan the 64-bit address space for virus code. Larry needed just one thing: earplugs to reduce the ventilation noise coming out of the strange box that he first mistook for an atomic reactor. But the earplugs were disposed of a long time ago, along with the beta boards, and nowadays it is the beautiful *Itanium2* that resides under Larry's desk.

While Larry did not care to open the guide for the *Itanium* instruction set, 'roy g biv' did. One question comes to mind: might 'roy g biv' have owned an *IA64* box? He probably did, but he could equally have used an emulator on a bulked up PC.

Intel's *IA64* assembly code is designed for explicit parallelism, so coding well in *IA64* assembly code requires the ability to 'think in parallel'. Unfortunately, when this is done well, the resulting code can be very difficult to read, especially when the virus code is further obfuscated. So, in turn, we started to think in parallel to share the fun of the analysis of this new kid on the block.

ROUND AND ROUND

The first time Rugrat is executed, it checks the event that caused its execution. The virus replicates only during the DLL_PROCESS_DETACH event, which occurs when an application is exiting. The reason that Rugrat does this could be because an application taking an extended period of time to terminate is far less noticeable than an application taking an extended period of time to start.

During Thread Local Storage events, it is NTDLL.DLL (the NATIVE API) that calls the Thread Local Storage entry point. That call leaves in the B0 register of *Itanium* processors a pointer into the NTDLL.DLL address space. The virus uses this fact to gain access to NTDLL.DLL, in order to retrieve the addresses of the API functions that it requires.

The virus uses a CRC method to match the API names. The use of the CRC method means that the API names are not visible in the virus code, while also reducing the size of the virus code.

The virus uses a few Win64 APIs from three different libraries: NTDLL.DLL, SFC_OS.DLL and KERNEL32.DLL. From NTDLL.DLL it picks LdrGetDllHandle(), RtlAddVectoredExceptionHandler() and RtlRemoveVectoredExceptionHandler(). The virus supports vectored exception handling to avoid crashing during infections. The use of LdrGetDllHandle() makes it simpler to gain access to other modules. From SFC_OS.DLL, Rugrat uses the SfcIsFileProtected() function to avoid infecting executables that are protected by the System File Checker (SFC).

The following 16 functions are used from KERNEL32.DLL to implement a standard direct action file infection of an *IA64* Portable Executable image using file mapping:

CreateFileMappingA()	GlobalAlloc()
CreateFileW()	GlobalFree()
CloseHandle()	LoadLibraryA()
FindFirstFileW()	MapViewOfFile()
FindNextFileW	SetCurrentDirectoryW()
FindClose()	SetFileAttributesW()
GetFullPathNameW()	SetFileTime()
GetTickCount()	UnmapViewOfFile()

As expected the virus will set the host's time/date stamp back after each infection, as well as its attributes, which it clears before infection.

Rugrat shows one major functional difference from Chiton – Rugrat uses only Unicode functions, and does not support ANSI functions, perhaps because 64-bit *Windows* is based on *Windows NT*, which is entirely Unicode under the hood.

The virus searches for files in the current directory and all subdirectories, using a linked list instead of a recursive function. This is important from the point of view of the virus author, because the .B variant of Rugrat infects DLLs, whose stack size can be very small.

FILTERS

Files are examined for their potential to be infected, regardless of their suffix, and will be infected if they pass a very strict set of filters. The first of these filters is the support for the System File Checker that exists in *Windows XP/2003 (Note the SFC_OS module name – while SFC exists in earlier versions of Windows the filename was changed in XP)*. The remaining filters include the condition that the file being examined must be a character mode or GUI application for the *Intel IA64* CPU, that the file must have no digital certificates, and that it must have no bytes outside of the image.

The *IA64* CPU introduces the concept of 'predication' to the execution flow, which allows a programmer to remove certain branches from the code, and to replace a block with a predicate check instead. A sample check for the file type might look like this:

```
      ld2 r30 = [r32]
      mov r31 = 0x5A4D;;
      cmp.eq p1 = r30, r31
(p1) ld4 r30 = [r8]
(p1) mov r31 = 0x4550;;
(p1) cmp.eq p1 = r30, r31
```

The first 'cmp' instruction sets the P1 register only if the condition is met. If it is not met, any instruction that is predicated by the P1 register will be ignored by the processor. The filtering code could be considered 'predication abuse', since it contains more than 30 predicated instructions in a row, including predicated compares, which reuse active predicate registers. The infection code contains several such blocks.

TOUCH AND GO

When a file that meets the infection criteria is found, it will be infected. If relocation data exist at the end of the file, the virus will move the data to a larger offset in the file, and place its code in the gap that has been created. If there are no relocation data at the end of the file, the virus code will be placed here. For the .A variant of Rugrat, which does not infect DLLs, the relocation data check is almost never used, since the majority of executable files do not contain relocation data (a 'global pointer' is used instead, see below). The .B variant of Rugrat infects DLLs in the same way as for applications, meaning that even 'resource-only' DLLs that have no main entry point can still be a source of infection, since the Thread Local Storage entry point will still be called.

The virus carries its own Thread Local Storage directory, which will be used if the target file contains no directory at all. The virus carries its own callback array for those hosts whose Thread Local Storage directory contains no callbacks. When it encounters a host that already has a Thread Local Storage directory containing callbacks, the virus will save the address of the first callback and replace it with the address of the virus code.

Once the infection is complete, the virus will calculate a new file checksum, if one existed previously, before continuing to search for more files.

When the file searching has finished, the virus will allow the application to exit by forcing an exception to occur. This technique appears twice in the virus code, and is an elegant way to reduce the code size, in addition to functioning as an effective anti-debugging method.

Since the virus has protected itself against errors by installing a Vectored Exception Handler, the simulation of an error condition results in the execution of a common block of code to exit a routine. This avoids the need for separate handlers for successful and unsuccessful code completion. The Vectored Exception Handling is a heap-based dynamic exception-handling mechanism of newer *Windows* releases, which provides an alternative to the Structured Exception Handling, a stack-based mechanism. SEH is more vulnerable to exploitation (see *VB*, September 2001, p.4) than VEH, and VEH has some other benefits such as up-front exception control. Nonetheless, the exploitation of VEH by attackers is also becoming common in the field.

EXCEPTION TO THE RULE

The cause of the exception is more subtle in Rugrat than in Chiton. On the x86 CPU, exception-causing instructions such as INT 3 can appear anywhere in the code, without restriction. On the *IA64* CPU, though, instructions are placed in 'slots', and collected in 'bundles' that execute in parallel, so an exception-causing instruction in a bundle that contains other instructions could cause those instructions to be interrupted. Additionally, when resuming from an exception handler, execution continues from the slot in which the exception occurs, which results in the instructions in the earlier slots of the same bundle not being executed.

These two restrictions are the likely reason why the virus author chose an instruction that is always placed in the first slot of a bundle. The instruction itself, LD1 R8 = [R0], also looks legitimate, until it is understood that the R0 register

always contains the value 0, and attempting to access the 0th byte of memory always causes an exception.

GLOBALISATION

The *IA64* uses a global pointer to access variables. This avoids the costly application of relocation items when a file is loaded to an address other than its default virtual address. This also makes the *IA64* code a little more compact, although it still appears large to eight-bit eyes.

The global pointer value can be retrieved from the Portable Executable GlobalPtr header field. Every structure – exported function addresses, Vectored Exception Handlers, even the Thread Local Storage Callback itself – is required to be in the form of a virtual address followed by a global pointer value, and the virus supplies these structures correctly. This structure is called a PLABEL_DESCRIPTOR, and the compiler and the debugger do a perfect job of hiding it, so guys like Larry need not worry about it.

CONCLUSION

The adoption of *IA64* machines appears to be slow. The lack of *IA64* machines restricts the scope of any potential outbreak, and this virus was simply a proof that it was possible. What can we expect next? A 32-bit and 64-bit cross-infector seems obvious.

Additionally, the news of *AMD64* is spreading, and systems are sold for as little as $700, "introducing the only Windows-compatible 64-bit processor and the smooth transition to 64 bits". *Microsoft* will be ready with the *AMD64* release by the end of this year. How much smoother could the transition be for the virus writers?

Chuckie: "So, we got a baby now."

Lillian DeVille: "I wished we'd a talked about it first. I don't know if I'm ready."

Rugrats, Klasky Csupo Inc.

Let's hope that this baby doesn't grow up.

W64/Rugrat.3344!IA64

Type:	Direct-action parasitic appender/inserter.
Infects:	*Windows IA64* PE files.
Payload:	None.
Removal:	Delete infected files and restore them from backup.

VIRUS ANALYSIS 2

PATRIOT GAMES

Gabor Szappanos and Tibor Marticsek
VirusBuster, Hungary

It is generally accepted that email worms know no geographical border, and spread all over the world within hours, showing homogeneous activity. But there are exceptions. One of these exceptions was Zafi.A: probably the fastest spreading email worm ever to be experienced in Hungary, but showing no impact at all outside the country.

To illustrate the impact of Zafi.A in Hungary, our email gateway captured about one virus per minute in the days before the appearance of this bug (this was in the middle of the Netsky-Bagle misery). This changed to the capture of one Zafi.A per second. Some impact. Even though it didn't appear until the second half of April, Zafi.A easily made it to the top of our virus prevalence list for that month.

As far as the number of infected messages is concerned, Zafi.A was the top virus in April 2004. Most of the messages are sent to non-existent email addresses, but in other gateways, which reject these messages, Zafi was still in the top five. In other countries this virus was not even noticeable.

At first glance, Zafi.A seems to be just another mass-mailer, however a closer look reveals some interesting details: this worm was definitely not written by a beginner. The worm code seems to be unfinished. More precisely, some code parts provide more functionality than is actually used by the worm. So further versions are to be expected. [*Zafi.B has appeared since this article was written and will be covered in detail in a future issue of VB - Ed.*]

ANALYSIS

The virus was written in assembly language, which is unusual for contemporary email worms. Even more unusual is the fact that there are three self-check calls in the code. These calls check three separate data areas by calculating a checksum of the appropriate string areas. If the areas have been changed (e.g. a message or a filename tampered with by a wannabe virus writer), Zafi aborts execution. This leads to the impression that the virus was written by an experienced assembly programmer. This assumption is further supported by the fact that some of the subroutines used in the worm provide a lot more functionality than is used by the worm. Either the author copied them from somewhere (but some of these subroutines are rather special), or the programmer was highly disciplined, going for a general solution. If the latter is true, we will hear from the 'SNAF team' (to whom the text in the worm's email message is attributed) in the future.

The first samples of Zafi.A arrived on the morning of 19 April 2004. The worm was written in assembly language, packed with UPX. It arrives in an email as an attachment, with a deceptive name:

```
link.matav.hu.viewcard.index42ADR4502HHJeTYWYJDF334
GSDEv25546.com
```

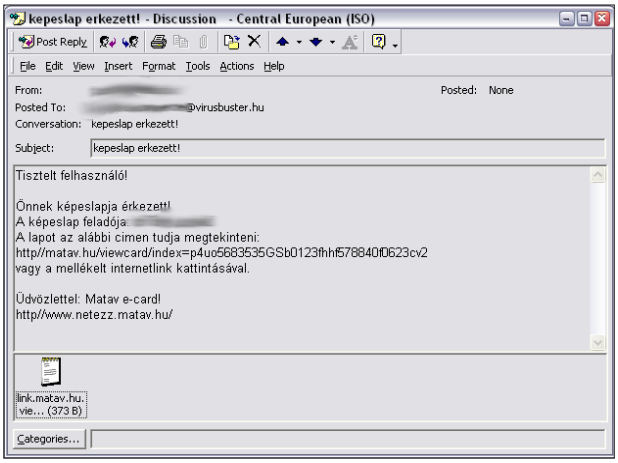
The worm spoofs the sender email address. The subject of the messages is:

```
kepeslap erkezett!
```

which translates: 'e-card arrived!'.

The message text (translated into English) reads:

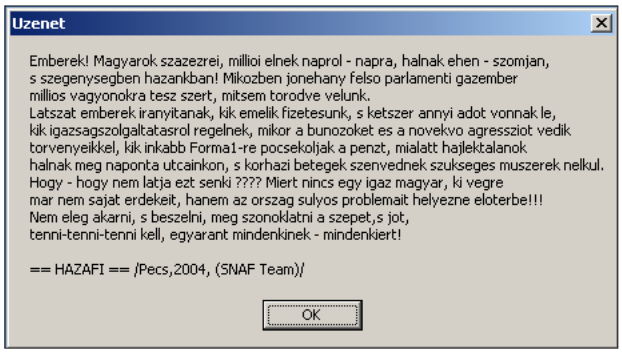
```
Dear User!

You have received a virtual card!

The sender of the card: {sender}

The card is accessible on the following address:

http//matav.hu/viewcard/
index=p4uo5683535GSb0123fhhf578840f0623cv2

or by clicking on the attached file.


Regards: Matav e-card!

http//www.netezz.matav.hu/
```



The message camouflages itself as a virtual greetings card, offering a link with the card. The link points to a non-existent web page. As an alternative, the virus offers the attachment to view the card – thus fooling the user into running the file.

The worm does not hide itself from the Task Manager list, and has no protection against running multiple copies of itself on the same computer.

When executed, the worm checks the system date. If the date is 2004.05.01, a MessageBox is displayed. Given the political attitude of the virus, this date was probably chosen because this was the day when Hungary became a member of the European Union. The message is political, a rough translation is as follows:




```
Folks! Hundreds of thousands of Hungarians are
starving and living in poverty. In the meantime, a
handful of bastards in the parliament are making
millions not caring about them.

These people are raising our salaries, while
taking away double in taxes, talking about justice
while protecting the criminals and the increasing
aggression with their laws; wasting money on Formula
1 while homeless people are dying on the streets
every day and the hospitals are in need of necessary
instruments.

How come nobody sees it? Why isn't there a righteous
Hungarian, who would take care of the problems of the
country instead of his personal interests?

It is not enough to wish and talk about the good, it
has to be done by everyone and for everyone.

== HAZAFI == /Pecs,2004, (SNAF Team)/
```

(HAZAFI translates to patriot, Pecs is a city in Hungary).

If the month is any other than April, the worm stops. If it is still April, the worm terminates the following processes:

zonalarm.exe	navdx.exe	nprotect.exe	fssm32.exe
vbsntw.exe	navstub.exe	ntvdm.exe	fsm32.exe
vbcons.exe	navw32.exe	ostronet.exe	fsbwsys.exe
pccguide.exe	nc2000.exe	vsmain.exe	fsgk32.exe
outpost.exe	ndd32.exe	vsmon.exe	dfw.exe
regedit.exe	netmon.exe	vsstat.exe	tnbutil.exe
regedit32.exe	netarmor.exe	vbust.exe	taskmgr.exe
navapw32.exe	netinfo.exe	mcagent.exe	winlogon.exe
pcciomon.exe	nmain.exe	fsav32.exe	fvprotect.exe

The virus code finishes with an endless loop which (waiting three seconds between each loop) calls the same routine to terminate these processes.

If the virus is executed without a command line (i.e. it is not executed via the startup registry key, where a random command line parameter is appended), it opens a randomly picked web page form the typed URL cache stored in one of the 'HKCU\Software\Microsoft\Internet Explorer\TypedURLs\url?' keys, where '?' is picked randomly in the 0..9 range.

The page is opened in the assumed default web browser, determined from the https web page assignment 'HKCU\Software\CLASSES\https\shell\open\command'.

If the selected key does not contain a location, the browser will not be opened.

The worm tries to create the registry key 'HKEY_LOCAL_MACHINE\Software\Microsoft\Hazafi'. Under this key it creates the following values:

"RA"=

set to dword:00000100 after email addresses have been collected.

"R1" =

"RegisteredOwner" (from registry key: Software\Microsoft\Windows NT\CurrentVersion or Software\Microsoft\Windows\CurrentVersion) if the former is not found.

"R2" =

"SMTP Email Address" (from Software\Microsoft\Internet Account Manager key)

"R3" =

path for created .exe virus name in the %System% directory.

"R4" =

path for created .dll virus name in the %System% directory.

"R5" =

path for the file in the %System% directory that contains the collected email addresses.

"R6", "R7", "R8" and "R9" may also store pathnames of email addresses, but are not used by the virus.

In the registry key 'HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run', the worm creates a value, for virus.exe file in the %System% directory. The registry entry has a random one- or two-character command line parameter appended, to avoid displaying a web page in the web browser during startup.

The worm creates a separate thread, which collects email addresses from files with the extension .htm, .wab, .txt, .dbx, . tbb, .asp, .php, .sht, .adb, .mbx, .eml and .pmr on C, D, E, F, G, and H fixed drives. It also extracts the addresses from the Windows Address Book, accessing it by the value of the registry key 'HKCU\ Software\Microsoft\WAB\WAB4\Wab File Name'.

It will reject addresses that contain the following substrings:

microsoft	f-prot	anti	avp	gov
vir	hotmail	panda	trendmicro	norton

As the worm's message is in Hungarian, it will avoid sending itself outside Hungary by rejecting the addresses that do not end with '*.hu*'.

It also rejects addresses that have the following 'domain parts':

lnk	dll	wav	wmv	gif	avi
swp	vxd	zip	cab	bmp	exe
ico	mp3	rar	pk3	mpg	jpg

This is most likely to discard the incorrect address-like filenames that were collected.

Apart from the collected addresses, the virus creates random usernames for each domain. These usernames are put together in pieces using a rather complicated algorithm from one- or two-byte fragments. The algorithm makes sure that the generated address is pronounceable (e.g. there are no consonants next to each other), and consists of between three and ten characters.

The first two characters are always selected from the list: vi, el, mo, ke, ka, en, ha, pa, sz, mi, ep, ho, em, cs, he, ko, ja, al, zs, ta, no, ad, os, or, pe, ut, po, ma, fi, am, fo, id, eg, fe, le, tu, gy, el, ki, jo, do, me, ny, to, ve, kr, ta, te.

After that the worm appends random characters in one to four steps. Before each step it checks whether the already generated address ends with a, e, i, o, or u. If not, it appends either one of a, e, i, o, u, or one of the fragments: ek, og, at, er, it, og, en, an, in, el, is, im, ik, ol, ak, eb, ad, az, as, ab, et, em, ed, ok, ot, nk.

If the already generated address ended with a vowel, the next part of the address will be selected randomly from either of the following lists:

ka, ra, va, za, be, re, li, di, mi, ca, ni, ti

or

b, c, d, f, g, h, j, k, l, m, n, p, r, s, t, u, v, z.

The efficiency of the algorithm is questionable, it was most likely borrowed from a spammer's code. It is very unlikely to generate a valid email address – most of the messages bounce at the email gateways.

Before sending the messages, Zafi.A checks whether the web page google.com is accessible (i.e. whether its name resolves). If no active connection is present, the worm waits for it in an infinite loop.

When it gathers the outgoing message, Zafi fills in the user name (from the "R1" key), the spoofed sender address (from the "R2" key) and the current date (if it fails to query it, it uses "04/20/04" instead).

At this point there is another self-defence measure: the worm checks the size of the DLL copy in the Windows directory (from the "R4" key). If it is anything other than 11,776 bytes (either a tampered-with or an unpacked sample is executed), the worm aborts the execution without sending itself.

The entire mail propagation code is then transferred to an allocated block in memory, and then a new thread is created and added onto this block. To make it work, the offsets of global variables in the code have to be fixed. This is done properly; the worm has a table of these offsets, and the offsets are relocated before the transfer of execution.

For each target address the worm tries to send itself via the target SMTP server, probing the following prefixes for each domain part:

fmx4.	relay.	domser.	suli2.	goliat2.
mail.	fmx.	fmx2.	mailb.	webmail.
fmx3.	mail01.	gate.	mail2.	postman.
smtp.	gold.	mx.	matav-4.	huasmtp01.
fmx1.	fmx5.	mx0.	www.	
matav-1.	matav-2.	mx1.	gemini.	

Some of those prefixes are widely used; others (matav-4, huasmtp01, matav-l, suli2) are specific to Hungary. If none of the servers work, the worm will not send the message to that address. Interestingly, it will not attempt to use the locally defined SMTP server or a predefined server. It might have been expected that this would slow its spread, but it seems that the prefix list was sufficient for the virus to work.

The worm uses its own SMTP engine to send the messages.

CONCLUSION

It is relatively easy to convince users in non-English speaking countries not to open messages in a language that they don't understand (well, not that easy, many of them will open them and run the attachments without understanding). But these users are completely unsuspecting if they receive messages in their native language. We learned this from the success of the Magold variants (see *VB*, August 2003, p.4) and Zafi (and what Germany experienced with Sober). The situation is further complicated by the fact that what we experienced as a huge outbreak in Hungary, was only experienced as a minor disturbance by the major AV vendors – thus they did not rush to release the updates as much as they would with a multi-country outbreak.

Many years ago local AV companies allegedly provided better protection against local viruses. Is this situation returning? I hope not. In Hungary, as in many other countries, the global AV companies have a large market share. In the case of these local threats, if the major players in the AV industry underestimate the threat, because it does not show up significantly on global levels, a large percentage of local users will be unprotected.

TECHNICAL FEATURE

GETTING INSIDE BEAGLE'S BACKDOOR

Michael Venable, Prashant Pathak, Arun Lakhotia
University of Louisiana at Lafayette, USA

This article presents a detailed analysis of a backdoor contained within the W32/Beagle.J worm (for a full analysis of W32/Beagle see http://www.virusbtn.com/resources/viruses/indepth/beagle.xml). The focus is on the capabilities and protocol of the backdoor, as well as the process used to uncover these capabilities.

Released in March 2004, Beagle.J propagates by emailing itself as an attachment to email addresses found in the address book and files of the infected host. The worm also spreads by placing a copy of itself in shared folders that are used by file-swapping programs such as *Kazaa*. The worm modifies the *Windows* registry so that it will run at each system startup. It even has code to remove itself from the infected system on or after 25 May, 2005.

The most interesting aspect of Beagle.J is the backdoor it opens. The backdoor allows an attacker to upload programs to an infected computer and execute them. These programs can be anything from simple keystroke loggers to more sophisticated backdoors, like kernel-mode rootkits, that provide complete dominance of the compromised machine.

As part of the 'Malware Analysis' course at the University of Louisiana at Lafayette, the students were given the task of discovering the protocol of the backdoor, which has not yet appeared in public, and were also required to write a client program capable of communicating with the worm. The results of this analysis are outlined in this article.

BEAGLE'S COMMANDS

The Beagle.J backdoor makes it possible for an attacker to remotely upload and execute arbitrary programs. The programs can be delivered in either of two ways: they can be uploaded from the attacker's computer or downloaded from a URL. Once on the victim's computer, the programs are stored in the victim's Windows folder (C:\Winnt, C:\Windows, etc.) and given the name 'iupld<x>.exe', where <x> is a randomly generated string.

The worm can execute a downloaded program either with no arguments or with the -upd argument.

In addition to uploading programs, the worm supports a command that tells it to remove itself from the victim's machine. Removal involves ending any running processes that belong to the worm and deleting the registry entry that causes the worm to run during *Windows* startup.

Altogether, the worm supports five different commands:

1. Upload and run a program.
2. Upload and run a program with -upd argument.
3. Download a program from a URL and run.
4. Download a program from a URL and run with -upd argument.
5. Remove the worm.

PROTOCOL

After successfully infecting a new system, the worm begins listening on port 2745. To communicate with the worm, a client connects to this port and sends a message containing three items: an initialization sequence (four bytes), a command number (one byte), and a password (up to 200 bytes, terminated by null). If an incorrect password is submitted, the backdoor will end the connection without sending any message back and without downloading the file. Assuming the password is correct, the worm sends an eight-byte response. At this point, the client can send any remaining data related to the chosen command. This process is illustrated in Figure 1.

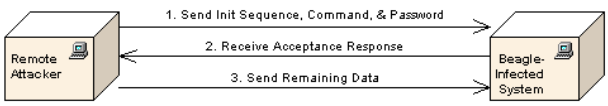


Figure 1. Communication between attacker and worm.

The initialization sequence is always 43 FF FF FF (hex) and the acceptance response received after submitting the password appears always to equal 03 00 00 00 B9 0A 00 00 (hex). The password, too, may be held constant (though multiple valid passwords exist), thus the only varying data is the command number and the remaining data related to the command. The following is a listing of each command and a description of the syntax of the remaining data.

Upload and run a program. To upload and run a program without the -upd argument, the command number should be 2. To run the program with the -upd argument, the command number 3 is required. When uploading a file, two pieces of information are needed as the remaining data. The first is a four-byte number in little-endian format representing the size of the file to be uploaded, and the second is the contents of the file.

Download a program from a URL and run. The command numbers for downloading a program from a URL are 8 (to run without -upd argument) and 10 (to run with -upd argument). In both cases, the remaining data should be a null-terminated string containing the URL from which to download the file.

Remove the worm from the victim's computer. The final option is to deactivate the worm. To do this, the command number needed is 4 and nothing need be submitted in the remaining data section. Hence, when comparing this command's syntax to Figure 1, step three can be omitted.

ANALYSIS OF THE BEAGLE.J BACKDOOR

We set up our analysis platform using the method described by Lenny Zeltser (http://www.zeltser.com/). The platform consists of a virtual environment package and a suite of static and dynamic analysis tools.

We used the virtual environment package available from *VMware* at http://www.vmware.com. This package is capable of running multiple operating systems, called guest systems, on top of the host operating system. The guest systems are capable of communicating with each other, but are isolated from the host operating system. By using this configuration, we are free to execute the malicious software without any significant risk of infecting production systems. *VMware* also contains a snapshot feature that makes it possible to save a (clean) state and then revert to that state should the system have been contaminated by the malware. We configured *VMware* with two operating systems: *Windows 2000* and *Red Hat Linux*. Figure 2 shows the setup.

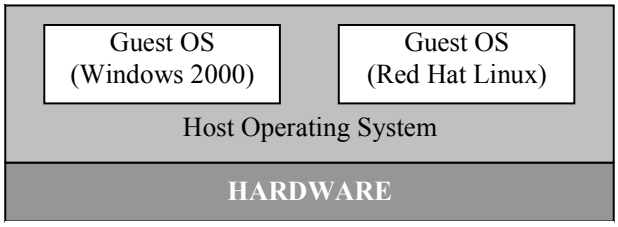


Figure 2. VMware configuration.

For the static analysis, we used tools such as a freely available UPX compression program available from http://upx.sourceforge.net/ to unpack the Beagle worm, which transmits itself in a compressed format. We used *BinText*, available from *Foundstone, Inc.*, to extract strings from the unpacked worm. For dynamic analysis, we used *Olly Debug*, a free debugger available online at http://home.t-online.de/home/Ollydbg. We also used tools such as *TCPView* and *RegShot*, available at http://www.sysinternals.com/ and http://regshot.yeah.net/, respectively. *TCPView* continuously monitors any network activity and *RegShot* monitors any registry and file access.

FINDING THE INITIAL BYTE SEQUENCE

We began analysis of the backdoor with the *Olly Debug* debugger. After setting breakpoints at all statements related

to network communication, we started the worm inside the debugger and initiated a connection to it from the *Linux* system via a simple client program capable of transmitting characters received from the standard input. Using the client, we were able to send characters to the worm and trace its execution to see how it processed the characters. We found that the worm checks the first four bytes for equality with 43 FF FF FF. If they do not match, the worm closes the connection.

We modified our client so that it sends the initial series of bytes correctly, allowing us to get past the first check performed by the worm. Using the same technique as above, we were able to deduce the next input expected by the worm, which is a single-byte number representing the command to perform. The worm checks to ensure that the command number is greater than 0 and less than 11. We modified our client again, choosing to send an arbitrary number within that range, since we did not know the exact command numbers at this point.

THE PASSWORD

The third check the worm imposes is the password. The worm reads 200 bytes or until the null character is read, whichever comes first. The bytes read make up the password. This password is then hashed and compared to the hash of the password. If the two hash values match, execution continues. Otherwise, the connection is closed. Using this technique, the author of the worm was able to avoid storing a plaintext copy of the password within the executable.

To assist in the analysis, we extracted the hashing function from the worm's assembly code and converted it to an equivalent C++ function. The new high-level hashing function is shown below.

```
void Hash(
 const unsigned char* buffer,   // IN
 int size,                      // IN: size of buffer
 int& r_eax                     // INOUT: register EAX
 int& r_edx)                    // INOUT: register EDX
{
 for(int i = 0; i < size; i++)
 {
   r_edx += buffer[i];
   if(r_edx >= 0x0FFF1)
     r_edx -= 0x0FFF1;

   r_eax += r_edx;
   if(r_eax >= 0x0FFF1)
     r_eax -= 0x0FFF1;
 }
}
```

When performing the hash, the worm calls this function three times. On the first call, the buffer is set equal to the password and r_eax and r_edx are initialized to 0xFFFF. For the second and third call, the buffer is 01 02 10 03 04 05 30 06 20 40 and 53 56 33 D2 33 DB 66 8B D0, respectively, and r_eax and r_edx are left unchanged from the previous call's return value. An example of the three calls is shown below.

```
r_eax = r_edx = 0xFFFF;
Hash( password, strlen(password), r_eax, r_edx );
Hash( "\x01\x02\x10\x03\x04\x05\x30\x06\x20\x40", 10,
r_eax, r_edx );
Hash( "\x53\x56\x33\xD2\x33\xDB\x66\x8B\xD0", 9,
r_eax, r_edx );
```

After the three calls, the worm validates the password by comparing r_eax to 0x9C02 and r_edx to 0x09C4. If the two match, the password is correct.

CRACKING THE PASSWORD

By carefully analysing the hashing function, we were able to derive two equations (for a complete derivation of the equations, see the appendix):

$$p_1 + p_2 + \ldots + p_{n-1} + p_n = 0x484$$
$$n \cdot p_1 + (n-1) \cdot p_2 + \ldots + 2p_{n-1} + p_n + n \cdot 0xE = 0x291E$$

where $1 \leq p_i \leq 255$ is the value of the i^{th} byte in the password and $n < 200$ is the length of the password in bytes.

We then constructed a program to pick each p_i randomly, such that the sum of all $p_i \approx 0x484$ and the sum of all $(n-i+1) \times p_i \approx 0x291E$, which matches the two equations discovered earlier. We chose to approximate the true values of the summation in order to decrease the time required for finding each p_i, as it would have been very difficult to guess the value of each p_i such that the sum equalled the value we needed exactly.

Within seconds, the program generated an approximation, and by manually adjusting the values of different p_i, we were able to determine a valid password for use with Beagle.J.

BRUTE FORCE PASSWORD CRACKING

While searching for a password using the above method, we were also tackling the problem from a different angle. We noticed that the hashing algorithm that is used maps an up to 199-byte password to a four-byte number. Thus, many different passwords can potentially be hashed to the same number, introducing a potential avenue of attack in our attempt to find a password.

To take advantage of this weakness, we constructed a program that would use a brute-force approach to cracking the algorithm, hoping that we would stumble upon one of the duplicate correct passwords. After running this program for several days, we realized it was unlikely to succeed and changed to a different approach. We modified the program

so that, instead of using a sequential brute force attack, it would randomly choose a large range of inputs and hash each value in that range to see if any hash to the right number. If none of the inputs in the range produced the correct value, a new range was selected. The idea was that by trying randomly chosen ranges of passwords, we would improve our chances of finding a valid password. Maybe this new approach was a good idea or maybe we were just extremely lucky, because after running for just a couple of hours, the program began to output a list of valid passwords.

THE COMMANDS

With the password out of the way, we were able to turn our attention to the commands. By stepping through the code with the debugger, we were able to uncover each of the valid command numbers: 2, 3, 4, 8, and 10. We analysed the code that reads input from the connection and determined the size and format of the expected input. By observing the function calls made by the worm, we were able to learn the purpose of each command.

THE CLIENT

Using the information obtained from the analysis, we constructed a client program capable of communicating with the Beagle.J worm. The client contains the initial byte sequence and the password hard-coded into the code, so the user only needs to supply the IP address of the beagle-infected machine and select a command. Using this client we were able to successfully execute all the commands on an infected machine in our laboratory environment.

CONCLUSION AND FUTURE WORK

Beagle.J's backdoor allows a remote user to upload and execute arbitrary programs. While, on the surface, this may appear limiting, it is actually all that an attacker needs to acquire control of a system. By uploading more powerful programs, the attacker can perform nearly any desired task, assuming the proper privileges are acquired. By understanding backdoors such as Beagle.J, we better equip ourselves with the knowledge needed to prevent such attacks in the future.

The analysis techniques we used consist of a combination of static and dynamic analysis. As viruses become increasingly easy to create and require less time to propagate, it becomes even more important for individuals to understand these analysis techniques and to know how to deploy them quickly. This paper presents one approach to doing so.

We have analysed another variant of Beagle, commonly known as Beagle.X. Like Beagle.J it also opens a backdoor but instead listens on port 2535. It has the same initialization, command, and password sequence as Beagle.J. But in this new variant, the set of instructions generated after validating the password is a function of the password itself. Hence, in order to execute the generated instructions correctly the original password is required. The author of the worm might have used this technique to make it harder for analysts to discover the password using a brute force approach. Our future work would involve analysing more viruses and worms to understand these kinds of intricacies. We would then develop techniques to counteract the viruses based on our experience.

APPENDIX

The following shows how to derive the two equations from the decompiled version of Beagle.J's hashing function. Assume the password is $p_1p_2\ldots p_{n-1}p_n$, where $n < 200$ and $1 \le p_i \le 255$. By tracing through the hashing routine, we see that after the first call to Hash, the values of r_edx and r_eax are:

$$r\_edx_1 = p_1 + p_2 + \ldots + p_{n-1} + p_n + E$$
$$r\_eax_1 = n{\cdot}p_1 + (n-1){\cdot}p_2 + \ldots + 2p_{n-1} + p_n + n{\cdot}E + E$$

The letter E is the hex number 0xE, written without the '0x' to reduce clutter. In the following discussion, all numbers are in hex and are obvious from context. Also, in order to keep the equations as simple as possible, we have assumed the conditional "if(r_eax >= 0x0FFF1)" will evaluate to true during the first iteration of the loop and never again. This is a valid assumption as long as $n < 17$.

After the second call:

$$r\_edx_2 = r\_edx_1 + B5$$
$$r\_eax_2 = r\_eax_1 + A{\cdot}r\_edx_1 + 234$$

Finally, after the third call:

$$r\_edx_3 = r\_edx_2 + 47D$$
$$r\_eax_3 = r\_eax_2 + 9{\cdot}r\_edx_2 + 136F$$

As stated, r_edx after the third call must equal 09C4, thus

$$\begin{aligned} r\_edx_3 &= r\_edx_2 + 47D = 09C4 \\ &\equiv r\_edx_1 + B5 + 47D = 09C4 \\ &\equiv p_1 + p_2 + \ldots + p_{n-1} + p_n + E + BF + 47D = 09C4 \\ &\equiv \mathbf{p_1 + p_2 + \ldots + p_{n-1} + p_n = 484} \end{aligned}$$

Similarly,

$$\begin{aligned} r\_eax_3 &= r\_eax_2 + 9{\cdot}r\_edx_2 + 136F = 9C02 \\ &\equiv r\_eax_1 + A{\cdot}r\_edx_1 + 234 + 9{\cdot}(r\_edx_1 + B5) + 136F = 9C02 \\ &\equiv n{\times}p_1 + (n-1){\cdot}p_2 + \ldots + 2p_{n-1} + p_n + n{\cdot}E + E + \\ &\qquad 9{\cdot}(p_1 + p_2 + \ldots + p_{n-1} + p_n + E + B5) + 136F = 9C02 \end{aligned}$$

But $p_1 + p_2 + \ldots + p_{n-1} + p_n = 484$, therefore

$$\begin{aligned} &\equiv n{\cdot}p_1 + (n-1){\cdot}p_2 + \ldots + 2p_{n-1} + p_n + n{\cdot}E + E + \\ &\qquad 9{\cdot}(484 + E + B5) + 136F = 9C02 \\ &\equiv n{\cdot}p_1 + (n-1){\cdot}p_2 + \ldots + 2p_{n-1} + p_n + n{\cdot}E + 72E4 = 9C02 \\ &\equiv \mathbf{n{\times}p_1 + (n-1)\,p_2 + \ldots + 2p_{n-1} + p_n + n{\cdot}E = 291E} \end{aligned}$$

FEATURE

ANTI–VIRUS SPAMMING AND THE VIRUS-NAMING MESS: PART 2

Dr Vesselin Bontchev
FRISK Software International

In the 'good old days' (i.e. until the mid-90s), when there were 'only' a couple of thousand known viruses, the anti-virus researchers responsible for maintaining the CARO Virus Naming Scheme used to get together at conferences, examine how each virus was named by the main anti-virus products, and agree upon what should be the 'proper' name for it. Sadly, nowadays, when there are probably more than 100,000 known viruses (and when nobody can tell you the exact number with a margin of error smaller than 10,000), this is no longer possible.

While tools like VGrep (for cross-referencing virus names across the different anti-virus products; see [1]) are a significant improvement on what we had in the old days, the glut of new viruses has overwhelmed our capabilities. It is simply unrealistic to expect that a standard agreement can be reached over the name of every single virus in existence.

A useful idea would be to reach an agreement at least on the names of the viruses that are known to be in the wild. However, this author believes that, as argued in [2], the so-called WildList is nothing of the sort. Worse, those who maintain it have consistently failed to stick to the existing *de facto* virus-naming standard for the viruses listed within it.

IMPORTANCE UNKNOWN

When a new virus appears, it is usually impossible to predict whether it will spread successfully. Certainly, a virus which spreads by email bursts is more likely to become widespread than a non-memory-resident COM-file-only infector. However, for every mass-mailing worm that sweeps the world, there are scores that never see the light of day outside specialized virus collections. When a new virus appears, it is impossible to determine immediately whether it will be one about which everybody will be talking (and, therefore, is worthy of an agreement upon a standard name) – or whether it will disappear into the virus collections among hundreds of thousands of other viruses, the names of which nobody remembers or cares about.

A solution to this problem would be to agree on a single standard name for every new virus. Unfortunately, the sheer number of new viruses means that this is no longer humanly possible. Alternatively, we could postpone this agreement until we know for sure that the virus is in the wild.

However, as mentioned previously, there is no reliable source of such information. Furthermore, it is often unacceptable to postpone such an agreement.

The reader should not be left with the impression that the situation is totally hopeless and that the anti-virus community is too stupid to do anything about it. The truth is that we *are* doing something about it. There are specialized forums (e.g. AVED, VCircle, VTech, etc.) where we report the new viruses which, in our estimation, are likely to become 'problematic' (i.e. widespread). We use these forums to send each other samples of the viruses, discuss what their names should be, and so on. The point is, that, too often, this is not sufficient to solve the naming mess.

RUSH TO PUBLISH

Gone are the days when a new computer virus took months to spread from one country to another. In fact, gone are the days of the Melissa virus, when they needed a couple of days to spread all over the world. Nowadays, viruses often need just hours (and, in rare cases, just minutes) to spread like wildfire from South Korea to the USA and back.

When such a virus appears, the anti-virus producers have very little time before the support phones start 'smoking' with calls from all over the world – both from people infected with the virus and from people who are just concerned about it and want to ask whether we detect it. In this short time, we have to:

1. Figure out how to detect the virus – this can be problematic if the virus uses extreme levels of polymorphism, entry point obfuscation, or weird tricks like spreading itself in password-protected archives or not residing in files at all (e.g. CodeRed).
2. Replicate the virus – this is often a non-trivial and time-consuming task. Viruses which have no problems spreading in the wild are often capricious and refuse to replicate in a controlled laboratory environment.
3. Figure out how to remove the virus – this is often much more difficult (although usually less urgent) than detecting it, especially when it involves precarious modifications to the Registry and some of the system DLLs. Sometimes a detailed analysis of the virus is required, in order to figure out how to perform this task properly.
4. Publish an update of our scanner that can handle the new virus properly. Sometimes publishing a detection-only update is good enough and proper identification and removal can be made available later.
5. Analyse the virus. Gone are the days of small, elegant viruses written in assembly language which were fun

to analyse. Nowadays the problematic viruses tend to consist of tens of kilobytes of some compiled obscure high-level language (often additionally compressed with some sort of executable file compressor).

6. Publish a description of the virus on our website and sometimes send an alert to our users.
7. Send a sample of the virus to other anti-virus researchers. In some cases this might be unnecessary – because, by the time we get around to doing it, the other anti-virus companies have already received a sample. Thousands of them, in fact.

Many of these tasks (e.g. detecting the virus, describing the virus) require that we pick a name for the virus. In the rush to perform all these tasks, we are forced to pick some acceptable name for the virus quickly. We simply cannot afford the time to discuss the issue with other anti-virus producers, to agree on a common name.

Of course, we are doing some things to alleviate this problem. The CARO Virus Naming Scheme gives a set of guidelines about how to pick the name for a new virus (and especially how *not* to name it). We have an informal agreement that the first researcher who sends a new virus to the rest of us under an acceptable name (i.e. one which does not violate the naming guidelines and which properly classifies the virus in the right family) has 'priority' – the other producers have to accept 'his' name and use it in their products. We run a set of scanners, in order to see whether there is a majority (or at least a high percentage) of products that already report the virus under a common name – or that recognize it as a variant of a known virus family.

However, in many cases, our best efforts simply fail. We sometimes have cases when several researchers pick equally acceptable but different names for the same virus and send them almost simultaneously. In the rush, we don't have the time to resolve what the proper name should be before we have to publish something that uses a name for the new virus.

One of the ideas that has been advanced to help alleviate this problem, is to have some kind of list of pre-approved names for future viruses – in the same way that meteorologists keep a list of names for the future hurricanes. When a virus appears, it should be given the name that is next in order on this list. In order to reduce the size of such a list and the work involved, this should be done only for the viruses that spread in the wild. Unfortunately, this idea has many deficiencies.

Hurricanes develop over multiple days and weeks, and there is plenty of time to name them. When a hurricane appears, there is little doubt that it is, indeed, a hurricane (or at least a major tropical storm) that is deserving of a name – whereas, when a new virus appears, we generally have no clue whether it will become widespread or not. Furthermore, nobody is likely to misuse a list of known future hurricane names for marketing reasons or for personal fame. However, if such a list of virus names is established and becomes public knowledge, the anti-virus companies will be under pressure from their marketing departments to pick from it the name of any obscure new virus they discover – because the mere fact that its name comes from the list will gain it (and, by association, them) wide press coverage. Likewise, virus authors are likely to design their viruses in such a way that naming them after one of the names on the list is the obvious thing to do – even if the virus doesn't become widespread.

DIFFICULT TO CHANGE

For many anti-virus producers, once they have picked a name for a virus in their products, it is extremely difficult (sometimes nearly impossible) to change it at a later stage. The original name is most likely already present in published press releases, in the virus description on the company's website and in the virus definition files for the scanner. If the name is suddenly changed (e.g. to the standard name agreed upon for that virus), there will be a lot of confusion among the users of that product. They will call the technical support department and ask why the virus is suddenly no longer listed as detected by the product or described on the website. These additional support issues, especially in the middle of a virus epidemic, are certainly something the anti-virus producers could do without!

In some circumstances, the original (inappropriate) virus name will already have been added to the printed manuals of the anti-virus product. Changing it would not only cause confusion and be difficult and time-consuming – but would also be rather expensive. For such reasons, at least some anti-virus producers have a policy never to change the name they have given to a virus.

THE STAND-APARTS

There are a few anti-virus producers who simply refuse to comply with a virus-naming standard. It does not matter what their reasons for non-compliance might be. If a standard is not followed, it cannot hope to be of much help.

LEVELS OF (IN)COMPETENCE

Another problem is that the anti-virus researcher who receives the new virus has to be sufficiently competent to implement handling (detection, recognition, identification and disinfection) of it – but does not necessarily have to be competent enough to name and classify it properly. Since

adding handling of many new viruses is the main thing that our virus labs do, we have invested significant efforts in simplifying the process and in training people who can perform this task.

With the current state of the art in our company, the author of this article can train a reasonably intelligent new employee how to implement handling of at least the simplest kinds of new virus in a couple of days. In contrast, learning how to analyse a virus and how to classify and name it properly requires years of experience and hundreds (if not thousands) of analysed viruses. There are few researchers with the required level of competence for this, and they might not be available when a new, explosively spreading virus appears – this is why virus names are sometimes picked by people who are not qualified for the task.

NO AUTOMATIC CLASSIFICATION

For most kinds of virus, we simply do not have the necessary tools for identifying and classifying them automatically. Macro viruses are a fortunate exception. We have excellent identification tools for macro viruses – in fact, these tools are so good that they allow us to share data that allows us to implement full handling of a new virus without even having seen it! By sharing such data on our internal discussion lists we can determine quickly whether a particular virus is a known one – and whether we're all talking about the same virus.

Furthermore, we have an excellent tool for classifying macro viruses. It uses a neural network, and while it cannot be relied upon blindly, it is often good enough to pinpoint the family in which a new variant should be classified, and even report the known variant in this family to which the new one is the most closely related. With such tools it is relatively easy to teach even a beginner how to classify and name macro viruses properly. Unfortunately, we do not have such tools for other kinds of virus.

WHAT'S IN A NAME?

The virus name that makes the most sense from a scientific point of view is not necessarily the one that is the most appealing or the easiest for the general public to remember. Consider the name 'virus://VBS/VBSWG.J@mm'. To a competent anti-virus researcher, this name is quite informative. It tells them that the malware is a virus, that it is written in VBScript, that it has been generated by the VBSWG virus construction kit, that this is the tenth virus generated by this virus-construction kit to have been discovered, and that this is a mass-mailing virus. For some unfathomable reason, the general public finds the name 'Anna_Kournikova' more appealing and easier to remember!

The virus-naming scheme in current use requires that we classify viruses into families, according to the code similarity of their replication part – so, we're forced to classify this virus in the VBSWG family. Of course, a reasonable question is: why didn't we name the family something less obscure than 'VBSWG' in the first place? The no less reasonable answer is that, when this virus-construction kit was first discovered, it was nothing more than yet another obscure, slightly buggy and, in general, remarkably unremarkable virus-construction kit. So, we gave it an obscure name and forgot about it. Nothing, at the time, indicated that the tenth virus generated with it would become a problem. We cannot afford the time and effort to pick memorable names for every new silly virus that appears.

Which leaves us with the problem that some anti-virus producers follow the agreed-upon virus-naming scheme and use the 'proper, scientific' name for the virus, while others use the name that is being used by the press and about which the users are likely to ask. Both approaches have pros and cons – but the end result is a virus-naming mess.

Worse still, new viruses are often detected automatically by some generic virus definition designed to handle a particular virus family. As a result, the scanners report these viruses as belonging to that family. In some cases this is a mistake – the virus should be classified into a different family. However, the producer of the scanner that detects the virus (albeit under an incorrect name) is usually happy that his product does so before anyone else's and is not in a hurry to change the reported name. In time, some products would choose to classify the virus properly – while others will use the 'incorrect' name under which the virus is already detected by another product. Sometimes, officially, the virus will be 'forced' into the (improper) known family, which creates a precedent and increases the virus-naming mess when new variants appear.

CONCLUSION

The aim of this article was not to provide any neat solutions. Its goal was simply to explain why the problems exist and to point out that, no matter how good a virus-naming standard we agree on, and no matter how well we design our products in respect of sending (or not) email warnings (see part 1 of this article – *VB* June 2004, p.9), these problems are likely to continue plaguing the users for the foreseeable future.

REFERENCES

[1] Project VGrep, http://www.virusbtn.com/resources/vgrep/index.xml.

[2] Vesselin Bontchev, 'The WildList – Still Useful?', *Proc. 9th Int. Virus Bull. Conf.*, 1999, pp. 281–287.

LETTERS

AV SPAMMING – A WORKABLE SOLUTION

Dr Bontchev is right to criticise the irresponsible spamming behaviour of some email scanning products, and email filters using anti-virus products (see *VB*, June 2004, p.9), but I think there is a workable solution in RFC2821. The problem arises because most scanners accept the message for delivery before they have examined the content. Thus, by RFC2821, they *must* send a notification if they are unable to deliver for any reason. When the receiver-SMTP sends the '250 OK' response to the completion of the DATA, it is accepting responsibility for delivery.

The solution is not to accept the responsibility until the message is known to be acceptable. Delay responding to the <CRLF>.<CRLF> until the anti-virus scanner has completed. If the message is infected, the response should be '550' instead of '250'. The sender-SMTP then knows that the message has been rejected, and has the responsibility of reporting the failure to the sender. If the sender-SMTP is, in fact, a virus, then it has gained the information that a server is protected, but how can it exploit that information? The challenge, particularly in a high-load environment, is whether scanning can be completed and the appropriate code returned before the connection times out. The simple store-and-forward approach to email scanning is obsolete, email anti-virus products must now become a real-time component of the email infrastructure.

Allan Dyer, Yui Kee Computing Ltd, Hong Kong

A LEAP IN THE WRONG DIRECTION

Rob Rosenberger's comment (see *VB*, June 2004, p.2) takes a *Sophos* press release and makes a logical leap in quite the wrong direction. *Sophos* is not recommending customers update their AV software 8,760 times a year as Rob suggests, but instead recommends that companies set their anti-virus software to *poll* to see if an anti-virus update is available on an hourly basis. Quite a different thing.

The vast majority of the time AV software polls to see if detection for new viruses is available it will be told that no update is required. But when an update has been made available it surely makes sense that the customer be given access to it sooner rather than later. Especially if later is the next scheduled update in, say, a couple of days time.

Combined with a small download size and no need for restarting updated computers, this is the appropriate way for AV vendors to protect customers when worms like Netsky can travel around the world in a matter of hours. *Sophos* agrees with Rob that AV updates alone are not the way to defend against viruses, and we continue to recommend businesses adopt a layered approach to their virus protection.

Graham Cluley, Sophos Plc, UK

PRODUCT REVIEW

GDATA AntiVirusKit

Matt Ham

GDATA's *AntiVirusKit* (*AVK*) has been reviewed numerous times by *VB*, and has an admirable track record over several years. During this period the scanning engines used by the product have varied – currently they are supplied by *SOFTWIN* (*BitDefender*) and *Kaspersky* (*KAV*). Products that use engines supplied by other companies can be divided roughly into two varieties: those which are in effect no more than a rebadge, with GUI functionality that is almost identical to that of the other company's product, and those which have clearly been constructed independently, making use of APIs from the engines but with no visual similarities. *AVK* is an example of the latter.

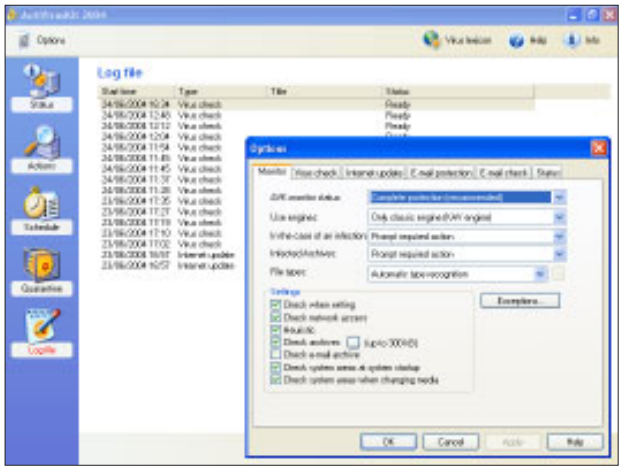
Although close to linguistic perfection in its English language version, *AVK* is most popular in Germany and *GDATA* is primarily a German company. Thus a much wider range of products is offered through *GDATA*'s German website than through the English-language versions of the site. However, the products of the *AVK* family are available on both sites, as is a selection of DVD-related software. The German site offers such additional applications as German-language technical dictionaries (for which, admittedly, the potential international market must be vanishingly small).

Versions of *AVK* are available for *Windows* platforms, SMTP and POP3-based mail gateways under *Windows* or *Linux* and both Samba and SendMail servers. On this occasion the *Windows XP* version was reviewed. Since the detection abilities of *AVK* were investigated in *Virus Bulletin*'s most recent comparative review (see *VB*, June 2004 p.12), these were not retested except in an ad hoc fashion when comparing different scanning modes.

INSTALLATION AND UPDATE

There are two main methods of installation for *AVK* – depending on whether an electronic or physical version is available. In the case of the physical medium, the process begins with a bootable *Linux* CD, with which the target machine can be pre-scanned for viruses. This is one of the more secure methods of determining that a machine is not infected prior to installation, though it is likely that many users will have downloaded the electronic version where this is not a feasible first step.

Installation of the electronic version is by means of an *InstallShield* application, thus being very familiar to those who have installed any significant number of PC applications during this century. The version used was 16 MB in size, with demo versions and German language



versions being of similar size. As expected, a licence agreement and installation location are followed by the choice of installation options: Typical, Compact and Custom. The Custom installation allows deselection of the on-access monitor component, right-click scanning, *Outlook* scanning, POP3 and IMAP scanning and the creation of a boot CD. At this stage there is no indication of how the Typical and Compact installations vary. The Typical installation option was selected for testing.

Next in the installation process are the options to disable weekly virus definition updates and a weekly scheduled scan. Neither is particularly advisable to miss in the real world, but for the purposes of testing the scheduled scan was disabled. After these options have been chosen, the installation process completes with file transfers, taking a negligible period of time and with no reboot required. Under *XP* a message bubble appears on the taskbar when the monitor is installed. Monitor activity is indicated by a shield and ball icon on the task bar; the monitor may be configured or simply disabled by right-clicking on the icon. The fact that the monitor is disabled is clearly indicated graphically – a small detail, but one which is often neglected.

In addition to the taskbar, program menu entries are added for the main scanner, manual, boot CD creation, uninstall, the *GDATA* website and the 'antiviruslab' website. Although the default URLs resolve to German language pages, there is a clear link to the English language versions ('International versions') from each of these areas. The application for boot CD creation may be used to create the physical medium described earlier, allowing for installation to and scanning of machines elsewhere in the organisation. This bootable CD should be of great use where machines are suspected to be infected.

Updates and upgrades are treated separately within the program, though the means of obtaining them is close to identical. When first updating, user information must be entered, including a registration number. When this information has been processed a login and username are available, to allow future updates and upgrades. Proxy servers are fully supported and where dial up connections require a username and password, these may also be stored here for ease of update.

WEB PRESENCE AND DOCUMENTATION

As mentioned above, an electronic manual is included with *AVK*, in addition to web-based resources. To complete the documentation, context-sensitive help is available within the main application.

The main website of *GDATA* is http://www.gdata.de/ – an extensive site in German. The English content is much less extensive and lacks a surprising amount of information – any form of downloads and support, for example. The resources here are limited to short product overviews, on-line purchase of the software and a company profile. Claims on the site that *AVK* holds a record for the anti-virus software with the most awards in the last 18 months may raise a few eyebrows elsewhere.

The usual additional security-related information is located at http://www.antiviruslab.com/. This contains virus descriptions, simple anti-virus advice, product demo versions and updates. When viruses are detected by *AVK*, the virus descriptions may be retrieved from here automatically for inspection. For machines without a direct Internet connection this is something of a problem – it would be a more useful tool if it could be redirected to a locally replicated version of the database. The reality behind such a simple convenience, however, might be too great or complex a task for simple implementation.

The manual is a very good example of a translated technical document which retains both meaning and readability. The details of day-to-day operation of *AVK* are described in detail, as is the use of a boot CD for installation. Steps within the *InstallShield* application are not completely covered by the manual, however, with the difference between the Typical and Compact installations remaining a mystery after having read through the PDF. The 'How To ..' section at the end of the manual will be particularly useful to new users of anti-virus software, while the manual itself remains clear and concise with little extraneous information. The PDF nature of the manual has been used well, with hyperlinks and occasional small graphics being used to aid in organisation and explanation.

The online context-sensitive help is very much a clone of this PDF manual, even having the same hyperlinks between related areas.

FEATURES

Akin to the manuals, the interface for *AVK* has a certain degree of simplicity and a lack of distractions in the areas where interaction will usually occur. If the main application is started with outdated virus definitions or without a recent check for new software functionality, this will be made clear – this warning can be disabled, although the process for doing so, a simple check box, is perhaps a little too simple.

Of note in the main GUI is the distinct lack of any drop down menu on the top of the interface and the presence of only one Options icon. Links are available to the help functions and virus descriptions site, and version information may be inspected, but the lack of duplicated view and control options is a welcome one. However, users who rely purely on a keyboard for navigating their scanner might be less impressed by this.

The views available – Status, Actions, Schedule, Quarantine and Protocol – are selected by the use of icons on the left-hand pane. The Protocol section is concerned with report files, the use of such a unique descriptor being one of the very few language oddities present.

Status is the default starting point, and gives an overview of current operation. The on-access scanner and email scanner status includes information as to which engines are operating. Program update and virus definition dates are also given here. The quarantine status and an overall Security/Performance rating are also displayed in this view. In each case double-clicking on the appropriate information enables the editing of related settings. As an example, the on-access scanner may be set from the default of scanning all files, to scanning only *Office 2000* files – the control here is not full by any means.

Of the Status settings, the Security/Performance area is perhaps the most interesting for the insight it gives into what measures may be used to streamline overheads. At the very lowest setting the *Kaspersky* engine alone is used – with only program data and documents scanned. Heuristics are still used on this setting. Increasing the security setting adds scanning of other file types, scanning on writing of data, scanning with both engines and scanning of archives (with various sizes scanned at different levels). These levels use predefined settings – though further tweaking is available via the Options icon.

The Actions view is that where on-demand scanning is performed. The areas for scanning are the usual selections offered – with updates and upgrades also accessible here though one click. Scanning over a network is fully supported.

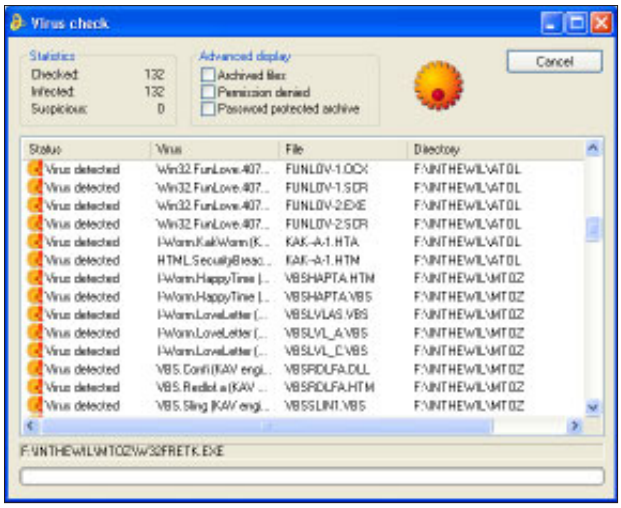
Schedule holds few surprises, the only item of note being that updates and upgrades may be scheduled here, with a weekly definition update being the default. This seems a little infrequent by modern standards and it is slightly odd that no application upgrade checks are made. Since this lack of scheduling will cause the upgrade warning to be triggered regularly, it would seem more sensible simply to settle for a default upgrade as well as update schedule. Quarantine and Protocol are even less remarkable.

A certain lack of control over the internal workings of *AVK* will have been noted so far, enabling the views to be remarkably simple to operate. Such control may be exercised through the Options icon, though this should not be needed by an average user. When invoked, the Options icon produces a set of tabbed interfaces, through which more detailed control may be exerted.

The tabs present cover Monitor, Virus check, Internet Update, email protection, E-mail check and Status. The features available here are mostly self explanatory. Of most interest are the three scanning mode controls – the Monitor, Virus check and E-mail check tabs, covering on-access, on-demand and email scanning respectively.

The greatest interest here lies in the treatment of objects, since two engines are available and heuristics may or may not be used, as the user sees fit. It is also possible to select all files to be scanned, scanning by file type or scanning only of subsets of potentially infectable objects.

This means there are a great number of possible permutations as far as scanning is concerned. For example, on-demand scanning may be instructed to use both engines optimised, both engines with most files checked twice, or either of the engines in isolation. Added to these choices, there are four choices for file types to be scanned, three for priority and for on/off selections available. This amounts to over 750 available combinations, with the reasons for making each choice not being within the grasp of an average user.



The centralisation of all these controls is something of a double-edged sword. Although the remainder of the application is certainly rendered simpler by the absence of any great choices except where to scan, the concentration of options in one place is likely to stun any hapless 'average user' who happens to stumble across them. Labelling them as 'advanced options' is not, in all cases, appropriate giving the developers something of a quandary as to where to place the Options icon. Its current position does not seem ideal, yet no better position springs to mind.

SCANNING

With such a plethora of scanning options available, there was considerable latitude for investigation of the effects of these on detection and on scanning speeds. Since both of the underlying engines are robust, however, the tests of throughput were considered more likely to be of interest.

First, the *VB* infected test sets were used for scanning. The default settings were used, with each change of scanning option being made in isolation. The options tested were: both engines using double-checking; only the *KAV* engine, only the *BitDefender* engine, no heuristics and scanning of all files. In each case logging only was selected as the action on infection.

The results were remarkable in that the differences were so small as to be negligible where throughput was concerned. With viral files it seems that the addition of an additional engine to the mix does not add appreciable overheads to the scanning process, even when double checks are specified. Likewise, the overhead of testing for file type (in a set where virtually all files are to be scanned) did not add an appreciable duration to the scans.

Where detection was concerned, all scans gave the same number of files scanned and detected, apart from the instance where the *BitDefender* engine alone was used for scanning. Here some files were detected by heuristics and a small number missed. Considering infected files alone, it would seem, however, that there is no advantage to be had in not using both engines.

The tests were repeated, this time on a clean install of *Windows XP Professional*. This was intended to supply a mix of file types, some infectable and others not, with a suitably real-world distribution. On this occasion the detection rates were all identical at zero, as would be expected. As far as scanning speeds were concerned, double-checking was marginally less speedy than optimised double scanning – though, taking only around 10 per cent longer, not to an appreciable extent. Removing heuristics from the equation also operated as expected, with the scanning in this case taking only 90 per cent of the original time.

Selecting only one engine, however, was the greatest change. Although the two engines were of similar speed in scanning, each was able to scan the files in some 60 per cent of the time taken for both in an optimised combination. Scanning all files, rather than selecting file type recognition, added 10 per cent to scanning time – less than might have been expected given the number of non-executable files present in a typical *Windows XP* installation.

CONCLUSION

As has been mentioned, *AVK* has been developed and documented remarkably clearly and concisely, making the product as a whole easy to use for a relative novice. There is also the degree of control available that will suit the purposes of a more advanced user.

How to exercise this control, however, is still a matter of some debate. While the standard settings of the software may be altered in many ways, most of these have little in the way of impact on performance. As a more paranoid user, for example, I would be tempted by the use of double-checking of all files, rather than the optimised scanning of actively detected file types. The debate comes over the decision of whether to use one or two engines, since this is the area in which there is the greatest impact on scanning performance. In the most recent *VB* comparative review either engine alone would have earned a VB 100% award for *AVK*. Neither *Kaspersky* nor *SOFTWIN* are infallible, however, and the double layer of protection afforded by the use of both is probably a good thing if a user's hardware can obviate the increased overheads.

Technical details

Product: *GDATA AntiVirusKit*.

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*.

Developer: *G DATA Software AG*, Konigsallee 178 b, D-44799 Bochum, Germany; email b-vertrieb@gdata.de; website http://www.gdata.de/.

ERRATUM: WINDOWS XP COMPARATIVE JUNE 2004

Due to a misinterpretation of the submittal procedures, an incorrect version of *SOFTWIN*'s *BitDefender* was submitted for testing in the June 2004 *Windows XP* comparative review. Retesting of the correct product resulted in a VB 100% award for the product, not the near miss as noted in that review.

END NOTES & NEWS

The Black Hat Training and Briefings USA take place 24–29 July 2004 in Las Vegas, NV, USA. See http://www.blackhat.com/.

The 13th USENIX Security Symposium will be held August 9–13, 2004, in San Diego, CA, USA. For details see http://www.usenix.org/

The National White Collar Crime Center's Economic Crime Summit takes place 17-18 August 2004 in Dallas, TX, USA. A federally-funded non-profit organization, NW3C has existed for the past 23 years to support state and local law enforcement efforts to prevent, investigate, and prosecute economic and cyber crimes. See http://www.summit.nw3c.org/.

The 19th IFIP International Information Security Conference (SEC 2004) takes place 23–26 August 2004, in Toulouse, France. Topics include intrusion detection, security architectures, security verification, multilateral security and computer forensics. For more information see http://www.laas.fr/sec2004/.

The High Technology Crime Investigation Association International Conference and Expo 2004 takes place 13–15 September 2004 in Washington, D.C., USA. The conference aims to provide training for all levels of the cyber-enforcement community from security specialists to law enforcement personnel. See http://www.htcia2004.com/.

The ISACA Network Security Conference will be held 13–15 September 2004 in Las Vegas, NV, USA and 15–17 November 2004 in Budapest, Hungary. Workshops and sessions will present the program and technical sides of information security, including risk management and policy components. Presentations will discuss the technologies, and the best practices in designing, deploying, operating and auditing them. See http://www.isaca.org/.

FINSEC 2004 will take place in London, UK on 15 and 16 September 2004, with workshops taking place on 14 and 17 September. Case studies and discussion groups will cover a range of topics including: Basel II/ IAS and IT security, prevention of online fraud and phishing scams, integrating technologies into a secure compliance framework, virus and patch management, and outsourcing IT security. For full details see http://www.mistieurope.com/.

The 14th Virus Bulletin International Conference and Exhibition, VB2004, takes place 29 September to 1 October 2004 at the Fairmont Chicago, IL, USA. For more information about the conference, including online registration, the full conference programme (complete with abstracts for all papers and panel sessions), and details of exhibition opportunities, visit http://www.virusbtn.com/.

Compsec 2004 will take place 14–15 October 2004 in London, UK. The conference aims to address the political and practical contexts of information security, as well as analysing leading edge technical issues. For details see http://www.compsec2004.com/.

RSA Europe takes place 3–5 November 2004 in Barcelona, Spain. For details see http://www.rsaconference.com/.

The 31st Annual Computer Security Conference and Expo will take place 8–10 November 2004 in Washington, D.C., USA. 14 tracks will cover topics including wireless, management, forensics, attacks and countermeasures, compliance and privacy and advanced technology. For details see http://www.gocsi.com/.

The 7th Association of anti-Virus Asia Researchers International conference (AVAR2004) will be held 25–26 November 2004 in, Tokyo, Japan. Those wishing to submit papers for the conference should do so before 30 June 2004. See http://www.aavar.org/.

Infosec USA will be held 7–9 December 2004 in New York, NY, USA. For details see http://www.infosecurityevent.com/.

Computer & Internet Crime 2005 will take place 24–25 January 2005 in London, UK. The conference and exhibition are dedicated solely to the problem of cyber crime and the associated threat to business, government and government agencies, public services and individuals. For more details see http://www.cic-exhibition.com/.

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 details email caroline.davison@nisc.org.uk or register your interest at http://www.nisc.org.uk/.

SUBSCRIPTION RATES

Subscription price for 1 year (12 issues) including first-class/airmail delivery: £195 (US$310)

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 www.virusbtn.com

vb Spam supplement

CONTENTS

NEWS & EVENTS

AOL EMPLOYEE COLLUDES WITH SPAMMERS

An *AOL* employee was arrested last month and charged with selling the company's customer email list to spammers. 24-year-old *AOL* engineer Jason Smathers is accused of stealing at least 92 million screen names from *AOL*'s database and selling the information to an associate, 21-year-old email marketer Sean Dunaway. Dunaway, who was also arrested, is accused both of using the screen names to promote his own business and of selling the information on to other spammers – an initial list for $52,000 and a subsequent updated list for $32,000.

AOL said it discovered the theft during an investigation it carried out as part of legal action the company was taking against another large-scale spammer earlier this year. A statement from the company read: "We deeply regret what has taken place and are thoroughly reviewing and strengthening our internal procedures as a result of this investigation and arrest."

The news comes only shortly after *AOL* – as one of the members of the Anti-Spam Technical Alliance – put forward a joint proposal aimed at reducing spam, which stated that spam cannot be stopped unless ISPs take responsibility for the problem. The Alliance recommends that all ISPs should run spam filters on outbound mail and prevent their customers from sending out more than 500 messages per day, or 100 per hour. Any suspicious accounts, the Alliance says, should be suspended immediately (as, one assumes, should employees found to be in cahoots with spammers).

SMS SPAMMER ARRESTED

Russian student Dmitry Anosov made history last month when he became the first Russian to be sentenced for sending spam – even though Russia does not currently have any anti-spam legislation. Anosov was found guilty of sending unsolicited (and 'unquotable') SMS messages to mobile phones. He has been put on probation for one year and ordered to pay a 3,000 rouble fine (approx. $100). Anti-spam legislation is still at the discussion stage in Russia.

7 STEPS TO A SPAM-FREE EXISTENCE?

Email security firm *Vircom* has issued a seven-step guide to avoiding spam, after its six-month study revealed (shock, horror) that responding to spam is the worst thing end users can do if they wish to avoid clogging their inboxes. *Vircom*'s SpamBuster team spent six months analysing the results of various levels of response to spam, from clicking the links in the messages, to sending back aggressive responses by return of message. The "stunning" results of the study showed that not only is there a dramatic increase in spam activity in accounts responding to spam, but "aggressive responses sent to spammers will actually trigger up to a 300 per cent increase in the junk you will get from them!" No kidding …

EVENTS

The ISIPP's International Spam Law & Policies Conference takes place on 29 July 2004 in San Francisco, CA, USA. For details see http://www.isipp.com/events.php.

The first Conference on Email and Anti-Spam (CEAS) will be held 30 July to 1 August 2004 in Mountain View, CA, USA. For more details see http://www.ceas.cc/.

A meeting of the ASRG will take place during the 60th IETF, which takes place 1–6 August, 2004 in San Diego, CA, USA. See http://asrg.sp.am/about/meetings.shtml.

Ferris Research will present a 'Webinar' entitled "Spam Vendor Shakeouts: Who's Surviving & Who's Leading", on 18 August 2004. The relative strengths and weaknesses of the leading anti-spam vendors and service providers will be discussed. See http://www.ferris.com/.

FEATURE

IT'S A SMALL (SPAM) WORLD, AFTER ALL

Terry Sullivan
QAQD.com, USA

One of the most hotly contested pieces of conventional wisdom regarding spam centres on the number of unique spam sources. Some authorities assert that "90 per cent of spam comes from 200 spam operations" [1], while other authors decry this as the "#1 Myth" regarding spam [2].

Since few spammers 'sign' their messages, it is impossible to achieve an accurate count of how many spammers are active at any given moment. However, the number of active spammers is at least broadly estimable based on approximately valid spammer-surrogates, such as the URLs contained within spam messages. Although such an estimate is necessarily indirect, and therefore somewhat imprecise, it is sufficient basis for a serviceable approximation.

Somewhat more precise, and potentially much more interesting, is the question of how spam is distributed among an arbitrary number of unique sources. An uneven distribution pattern would imply that, even if the number of spammers were 'large' (arbitrarily defined), then a disproportionately large amount of spam must necessarily originate from a disproportionately small number of sources.

Thus it is possible, in some sense, to idealize the problem, and imagine two extreme scenarios regarding the origins of spam. At one extreme, the distribution of spam is approximately uniform, with each spammer accounting for a roughly equal percentage of total spam sent. In this scenario, having a precise estimate regarding the number of spammers is critical to any analysis. At the other extreme, a comparatively small number of highly prolific spammers accounts for the bulk of spam received, thus mitigating the need for a precise estimate regarding their exact number.

Ultimately, three distinct lines of evidence all strongly support the conclusion that the number of unique spam sources is indeed relatively small (numbering in the hundreds, not thousands) and that only a few dozen spam sources account for the vast majority of spam worldwide.

URL FREQUENCY

Most spam messages are directed at motivating the recipient to visit a website to place an order. Thus, the URLs advertised in those messages, and the domain names in particular, serve as a potentially fruitful and approximately valid surrogate for spam source. While no one expects an exact one-to-one correspondence between domain name and spammer, it is reasonable to expect the distribution of 'spamvertised' URLs to mirror approximately the distribution of spammers.

One of the more intriguing recent entries into the anti-spam movement is Jeff Chan's *Spam URI Realtime Block List* (*SURBL*) [3]. Among its various data sources, *SURBL* extracts domain names from URLs contained in messages submitted by *SpamCop* users. Any domain with at least 20 individual reports is included in the *SURBL* block list. The inherent diversity of reporting sources helps both to ensure breadth of coverage and to minimize systematic sampling bias. Recently, *SURBL* has initiated a 'rollup' procedure that all but eliminates the effects of spammers' inclusion of random subdomains within the URL.

For analytical purposes, some 326 domain names meeting the *SURBL* inclusion criteria during a four-day 'window' in the second week of June 2004 were examined. The most striking thing about the distribution of domains is that it is profoundly non-uniform. The distribution of spam domains exhibits a marked power-law characteristic. Power-law distributions (most often known as Zipf or Pareto distributions) share one feature in common: the product of frequency (in this case, spam volume) and rank (most-prolific to least-prolific) is approximately constant. A log-log plot of frequency-by-rank describes an approximately straight line.

Figure 1 shows just such a log-log plot of the *SURBL* data, confirming an approximately linear relationship between frequency and rank. While that relationship is not perfectly linear, the distribution of spam URLs in no way resembles a uniform distribution (shown in the figure as a dotted line). In this sample, barely five per cent of the URLs account for over 25 per cent of spam messages reported, and less than 20 per cent of the URLs, numbering just a few dozen in all,

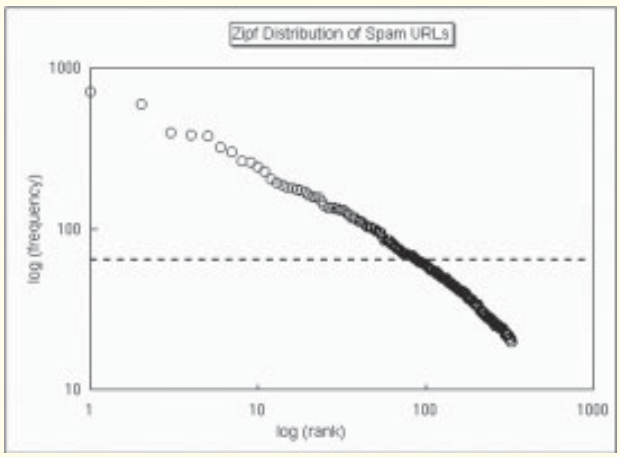


Figure 1: Log-log plot of the SURBL data, confirming an approximately linear relationship between frequency and rank.

account for over half the total spam. To the extent that many of these domain names are simple variations on a single 'root' name, the number of actual spammers is almost certainly smaller still.

'SMALL-WORLD' PATTERNS

When two strangers meet at a party, and discover during the course of conversation that they share a friend in common, it is commonplace for one or both to exclaim, "Wow, small world!" In reality, there is a less exotic explanation: the two 'strangers' are members of a common extended social network (which explains how they both came to be invited to the same party). Discovering 'small world' connections within such locally-organized networks is dramatically more likely than within a network of truly random connections.

It is not surprising that 'small world' patterns are commonplace in spammer behaviour and activities. Consider as an example: domains A through M are all bulk registered on the same day, through the same registrar, by the same registrant, and all are subsequently used to advertise a single product. The obvious (and almost certainly correct) inference is that all of this activity originates from a single spammer. Just a few days later, domains N through Z are similarly bulk registered together (perhaps via a different registrar) and all are used to advertise a different product. Taken at face value, this pattern suggests a maximum of two spammers.

But now imagine that the 'whois' data for the registrant of domains N–Z points to an email address in domain_A. Even if the address itself is bogus, the use of domain_A requires knowledge of the existence of domain_A. Thus, two highly clustered but seemingly unconnected sub-nets 'join' to form a single, interconnected whole. Such examples of connectedness between clustered nodes is common, even ubiquitous, among domain names used in spam.

FEATURE EVOLUTION

The 2004 MIT Spam Conference included an empirical study of the evolution of spam features [4]. Perhaps the most striking result of that study is that changes to spam features strongly resemble 'punctuated equilibrium' from evolutionary biology. When analysed in aggregate, spam features remain remarkably consistent for months at a time, but then are subject to sudden and dramatic change. (These results have since been replicated multiple times, with different feature sets, different time windows, and different test corpora. Although small fluctuations in the exact values of the observations inevitably occur, the 'punctuated equilibrium' phenomenon remains unchanged.)

A sudden, wholesale shift in spam features is utterly incompatible with a large number of spam sources. It is difficult to imagine how a large number of spam sources would independently and simultaneously come to alter their tactics in virtually identical ways. However, such a wholesale shift is entirely consistent with a relatively small number of spammers, and the observed power-law distribution of spam origin. Stated differently, if a few spammers are responsible for the majority of spam, then a shift in tactics among a few individuals will inevitably result in a sudden, large variance in spam features.

CONCLUSION

Arguably, none of these lines of evidence alone is sufficient to support a definitive inference regarding the number of spammers. When taken together, however, these three sets of data converge on a single conclusion: that the number of spammers worldwide is at most a few hundred, and most spam originates from a maximum of a few dozen highly prolific sources. For the observed results to reflect a uniform distribution among a large number of sources, it would be necessary to posit extraordinary coordination (and amazingly effective enforcement) among a diverse, diffuse, globally distributed group. These results are consistent with the patterns predicted by an uneven distribution of spam, originating from a relatively small number of unique sources.

These results have specific implications for the fight against spam. The uneven distribution characteristic of spam suggests that great benefits may be obtainable from tightly focused anti-spam efforts that specifically target the most prolific sources of spam. Technologically, these results help to explain the disproportionate success of a computationally simple project such as *SURBL* – and suggest that similarly focused legal remedies, whether civil or criminal in nature, may also prove effective. Finally, these results illuminate potentially fruitful avenues for technology R&D efforts. In particular, robust author-identification technologies have the potential to provide broad support to both technical and forensic efforts in the fight against spam.

REFERENCES

[1] Registry of Known Spam Operations, http://www.spamhaus.org/rokso/.

[2] The 10 Biggest Spam Myths, http://www.clickz.com/experts/brand/buzz/article.php/3112021.

[3] Spam URI Realtime Block List, http://www.surbl.org/.

[4] The Myth of Spam Volatility, http://www.qaqd.com/research/mit04sum.html.

SUMMARY

ASRG SUMMARY: JUNE 2004

Helen Martin

The start of this month marked the end of an era as Yakov Shafranovich announced that, due to increasing time constraints, he would be stepping down from his position as ARSG co-chair. This leaves John Levine as sole ASRG chair. In his final message as co-chair, Yakov posted a link to the CallerID Internet draft, http://www.ietf.org/internet-drafts/draft-atkinson-callerid-00.txt.

Phillip Hallam-Baker drew the group's attention to news of a spammer who has been given a seven-year prison sentence for sending 850 million unsolicited emails. Phillip felt that this case would be the first of many and said that, while this would not discourage existing spammers, it might help to slow the rate at which new spammers come to the fore.

Having declared himself to sit firmly in the camp that believes that the digital signing of the entire bytes of an RFC2822 message body is too fragile an approach to be of value, Bob Atkinson revealed his own draft document for email postmarking. In his own words, this is a draft for "non-user-level signing of email that supports the ability to affix domain-related or other signed information to a message while taking pragmatic steps to be robust in the face of transformations that occur to messages as they flow in the Internet." A long discussion on the proposal and on the ins and outs of email signing followed. Bob's draft can be found at http://www.lessspam.org/EmailPostmarks.pdf.

Jim Fenton pointed the group to a new Internet Draft for a message-signing protocol, not dissimilar to DomainKeys. Like DomainKeys the body and selected headers of the message are signed, with the signature appearing in the message header. However, in this case the public key associated with the signature is included as part of the message and the authorisation of that key is checked with an element called a Key Registration Server found through the DNS. The full draft can be viewed at http://www.ietf.org/internet-drafts/draft-fenton-identified-mail-00.txt.

Jeff Silverman has been getting his family in on the act – he reported that, while on vacation, his father came up with a proof that it is not possible to build a 100 per cent effective spam filter based purely on the content of messages. Jeff concluded that, since it is impossible to solve the spam problem by content analysis, some sort of white/black/grey list solution is required. der Mouse questioned the mathematical validity of Mr Silverman's proof, but concurred with Jeff's conclusion because, in some cases, "the only difference between a spam and a ham lies in the domain of human intent and consent, which are not part of the content of a message." Other respondents indicated that, while 'spaminess' cannot depend only on the message content, the content analysis that is currently deployed has its place, and that, ultimately, the solution is a broad combination of mechanisms (including white/black/grey lists) which will force up the cost of spamming.

Barry Shein had read the FTC's report on the (non)viability of a national 'do not email registry' (see http://www.ftc.gov/reports/dneregistry/report.pdf) and found it to be a well-written summary of current thinking about spam and spammers and worthy of recommendation for anybody wanting to get a good idea of "what the nature of the spam beast really is." However, he was disappointed by what he felt was the document's "defeatist tone" and disagreed that message authentication is what is needed, laying the responsibility instead at the feet of the ISPs – "those who already know who is using their resources".

After testing the water for potential interest, John Levine announced a new subgroup for Identity, Accreditation, and Reputation (iar@asrg.sp.am). The first item of business for the group will be to create a short charter – John was keen to point out that the goals of the group are not to build an IAR system, but to work out what current IAR systems have in common as a basis for standardisation.

Markus Stumpf provided some food for thought with a post relating to the speed of deployment of anti-spam strategies. Markus had asked Peter Koch (who compiles a regional hostcount of DE domains for *RIPE*) for some figures – specifically, a count of MX records normalised to unique IP addresses. The numbers were: 7,600,000 second-level DE domains, compared with 144,582 genuine IP addresses used as MX hosts. Markus made the point that, assuming that MX hosts and MTAs are of the same order of magnitude, any anti-spam strategy that attempts to authorise MTAs with records in domain zones will be considerably more time-consuming than a scheme that manages to authorise MTAs *per se*.

Philip Miller noted that, while the *New York Times* was running an article reporting that the 'big 4' ISPs (*AOL*, *Earthlink*, *MSN*, *Yahoo!*) are lining up to test SPF, CallerID, and DomainKeys by the end of this year, the article made no mention of the IETF or the MARID working group that is moving these specifications forward.

The final quote this month is from Andreas Saurwein, whose advice was: "*Never* underestimate what people are able to believe. Otherwise we would not have people sending emails to Bill G. in the hopes to receive US$ 50, or to cellular handset manufacturer X to receive their brand new free cell phone ... or reply[ing] to spam."

[*As always, an archive of all ASRG postings can be found at http://www1.ietf.org/mail-archive/web/asrg/current/.*]