JULY 2009

virus BULLETIN

Fighting malware and spam

CONTENTS

IN THIS ISSUE

SQUATTERS' RIGHTS

Typosquatting takes advantage of the typographical mistakes often made by users when entering a website address into a web browser. Amit Verma discusses a two-step approach to combatting the problem, prioritizing the registration of domain typos and detecting typos entered into Internet browsers and email clients.

page 15

ON PATROL

Since 2005, the Malware Patrol Project has been cataloguing URLs used in phishing scams and distributing block lists for the most popular proxies and anti-spam systems. André D. Corrêa describes the challenges of collecting and monitoring malicious URLs.

page 18

ANTI-SPAM CERTIFICATION

In VB's second round of anti-spam comparative testing and certification the all-important question was whether the high achievers from the first test could maintain the same high standards this month. Martijn Grooten has the results of a test in which more products were tested against a larger spam corpus and with stricter benchmarks.

page 25

ISSN 1749-7027

virus BULLETIN **COMMENT**

'The intent is the same, the information displayed to the user is the same, and the extorted money probably ends up in the same pocket.'

Pierre-Marc Bureau, Eset

SAME MALWARE, DIFFERENT CODE

As Dijkstra once said, 'The effective exploitation of his powers of abstraction must be regarded as one of the most vital activities of a competent programmer.'[1] Malware operators and their sponsors understand this advice and are pushing the concept of abstraction to the next level. The gangs profiting from malware attacks do not really care about programming languages or anti-debugging tricks; they are interested in software that is relatively bug free and that matches their requirements (enabling them to steal passwords from game XYZ, send spam, etc.). Sometimes we need to take our heads out of the petri dish and stop over-analysing the programming detail. In more than one sense, the point isn't how the code works, it's what it does that matters.

Many of us are aware of the Waledac family of malware, which has now been around for several months. This threat is used to send massive amounts of spam. It communicates using a peer-to-peer network and spreads through malicious websites that use fast flux DNS server entries in order to make blocking by IP address more difficult. Does that description sound familiar? Exactly the same words could have been used to describe the Storm worm. This highlights an interesting problem we are facing increasingly frequently: similar malware, similar operations, but a completely different code base.

[1] Dijkstra, E.W. The Humble Programmer, 1972.

Editor: Helen Martin
Technical Editor: Morton Swimmer
Test Team Director: John Hawes
Anti-Spam Test Director: Martijn Grooten
Security Test Engineer: Simon Bates
Sales Executive: Allison Sketchley
Consulting Editors:
Nick FitzGerald, *Independent consultant, NZ*
Ian Whalley, *IBM Research, USA*
Richard Ford, *Florida Institute of Technology, USA*

Rogue anti-virus programs have been prolific in the last 12 months. They are usually installed by other malware and generate income for their operators by scaring users and leading them to believe that the only way to clean their computer is by sending money to an unknown company. Once again, we have identified dozens of different code bases for rogue anti-virus programs. Some of them are programmed in Visual Basic, some in C++, Delphi, and so on. The intent is the same, the information displayed to the user is the same, and the extorted money probably ends up in the same pocket.

It has become clear that the organizations behind malware operations are prepared to sponsor complete rewrites of their malware. This may be to repair previous programming errors or design limitations, but they are also doing it to keep one step ahead of the research community and to evade anti-virus detection. It takes days, if not weeks, for a skilled reverse engineer to analyse and understand a piece of malware completely. Thus deploying completely new code will slow down investigative work considerably.

We need to take a step back from simply looking at compiled code. We have to focus on finding a model that will express program functionalities and intents. By deducing and classifying the intent of a program (or at least part of it), we might have a chance of identifying the relationship between malicious programs that are part of the same operation and built for the same purpose but which use different code.

This task will be very hard to accomplish and we cannot expect flawless results, but any effort in this area would be a step in the right direction. For example, Dullien and Porst[2] have developed a platform-independent language to represent disassembled code. While this approach gives results that are too detailed to be effective in creating a generic description of a program's intent, it does show that independent representation is attainable. The next step is to create a model that is easily extractable from compiled code and reliable enough to recognize similarities in the intent of the gang using variable binaries as part of their operation. One way to develop this model would be to include both algorithmic information from the code and data it is using as parameters.

It follows logically that it will be even more difficult to decide whether two programs using different code bases have the same intent. Yet in practice we can (and do, almost routinely) deduce malicious intent behind unknown code. We miss a trick or two by focusing on the crime scene and forgetting the criminal.

[2] Dullien, T.; Porst, S. REIL: A platform independent intermediate representation of disassembled code for static analysis, 2009.

NEWS

SPAM AWAY

Observant readers will notice the absence in this month's issue of the *VB Spam Supplement*. Almost six years after its introduction the *Spam Supplement* has been retired because the distinction between spam and malware is becoming increasingly blurred and there seems little reason to continue to segregate the subjects. *VB* will continue to cover spam and anti-spam issues, but from now on these articles will be integrated into the rest of the magazine.

JOBS FOR THE NAUGHTY BOYS

Lord West, the UK's Parliamentary Under-Secretary for Security and Counter-terrorism, raised the hackles of security experts last month when he outlined some of the steps taken to beef up the UK's cyber defence strategy. Lord West claimed the UK was ahead of the rest of the world in its cyber security strategy, and went on to reveal that the government had recruited a team of former hackers for its new Cyber Security Operations Centre. In an attempt to justify the appointments, he said that the government had not employed any 'ultra, ultra criminals', but would be calling upon the expertise of former 'naughty boys'.

Commentators across the security industry have questioned the wisdom of employing former script-kiddies (a group whose technical abilities have never been hailed as anything other than flaky) and giving them responsibility for national security. What appears to be lacking in the cyber defence strategy outlined so far is any attempt to harness the expertise of those who really know what they are talking about – seasoned computer security professionals.

SPAMMER TO SERVE TIME

Prolific spammer Alan Ralsky is facing up to 87 months in jail after pleading guilty to his part in a pump-and-dump stock spam scam. Ralsky and his son-in-law are believed to have masterminded the scam, in which spam was used to artificially boost interest in Chinese penny stocks – the scammers would then trade in their shares when the value of the stock reached its highest point. Ralsky pleaded guilty to conspiracy to wire fraud, mail fraud, money laundering and CAN-SPAM offences. He and his accomplices are estimated to have made $3 million from the scam in just 18 months.

Ralsky reportedly once admitted sending more than 70 million messages a day, and in 2003, just prior to the CAN-SPAM Act coming into force, he said that the passage of the US bill through the House of Representatives had 'made [his] day' – referring to what was considered the relatively lax nature of the bill which put the onus on the user to opt out of mailings. No doubt anti-spam campaigners will be delighted that Ralsky has finally had his come-uppance.

Prevalence Table – May 2009

Malware	Type	%
Dropper-misc	Trojan	19.03%
Waledac	Worm	17.95%
Agent	Trojan	13.71%
NetSky	Worm	8.30%
Invoice	Trojan	6.94%
Virut	Virus	5.97%
OnlineGames	Trojan	2.73%
Iframe	Exploit	2.61%
Mytob	Worm	2.59%
Mydoom	Worm	2.05%
Downloader-misc	Trojan	1.83%
Basine	Trojan	1.49%
Heuristic/generic	Misc	1.47%
Suspect packers	Misc	1.14%
Inject	Trojan	1.06%
VB	Worm	0.88%
Bagle	Worm	0.81%
Zlob/Tibs	Trojan	0.74%
Delf	Trojan	0.68%
Lineage/Magania	Trojan	0.66%
Mabezat	Virus	0.52%
Zbot	Trojan	0.47%
Small	Trojan	0.45%
Alman	Worm	0.44%
Zafi	Worm	0.42%
LDPinch	Trojan	0.41%
Autorun	Worm	0.40%
Edibara	Trojan	0.38%
Sality	Virus	0.36%
Cutwail/Pandex/Pushdo	Trojan	0.31%
Heuristic/generic	Trojan	0.29%
Encrypted/obfuscated	Misc	0.26%
Murlo	Trojan	0.24%
Others[1]		2.37%
Total		**100.00%**

[1]Readers are reminded that a complete listing is posted at http://www.virusbtn.com/Prevalence/.

MALWARE ANALYSIS 1

CAN YOU SPARE A SEG?

Peter Ferrie
Microsoft, USA

Peter Ferrie resumes his series of analyses of viruses contained in the EOF-rRlf-DoomRiderz virus zine (see also VB, September 2008, p.4, VB, October 2008, p.4 and VB, November 2008, p.4).

NON-OPTIMIZATION TRICKS

We begin with a virus that was named 'H2T3' by its author. This virus infects files on the *FreeBSD* platform. Interestingly, the virus is split into two parts. The first part is written in assembly language, and exists solely to pass some important constants to the second part, which is written in C.

The assembly language part is not optimized at all. For example, a MOV and an ADD could be replaced by an LEA; some arithmetic involving two constants could be achieved with one combined constant, etc. Even the calling convention that was used in the first part results in an extra instruction to balance the stack, but this is perhaps an indication of the quality of work by virus writers these days. It is unclear even why the first part exists, since the constants could be calculated just as easily in C.

Seek and ye shall find

The virus begins by searching for regular files within the current directory. For each file that is found, the virus attempts to retrieve the file attributes and change them to writable. The file is skipped if either of these operations fails. If both operations succeed, then the virus attempts to open and map the file. If the open fails, then the virus restores the file attributes and returns. If the mapping fails, then the virus attempts to unmap an invalid region – fortunately for the virus writer, this invalid unmapping does not cause an error.

However, the virus is extremely trusting of the contents of the file. It assumes that the file is in ELF format before proceeding. The assumption goes so far that a field inside the supposed ELF header is used by the virus, without checking that the file is large enough to support the presence of that field. A sufficiently small file will cause the code to crash. In fact, a truncated ELF file, or a file with a sufficiently large value in the e_phnum field, among other things, will cause the virus to demonstrate the same effect, since the code contains no bounds checking of any kind.

Of course, these are minor quibbles.

Image-conscious code

The virus is interested in ELF files that are executable, not infected already, and whose ABI specifies a *FreeBSD* file. The virus does not check the target CPU for the file, perhaps assuming that any file on the current system is designed to run on that system. The virus then searches within the Program Header Table entries for all loadable segment entries, and keeps track of the one with the lowest virtual address. This value is used as the ending address for the virus code in the file to infect. What the virus intends to find is the entry with the physical address of zero, which is the file header, and which corresponds to the image base address. The virus is simply performing the search in a different way.

Headers and footers

The virus also searches within the Program Header Table entries for a PT_PHDR (Program Header Table segment) entry. If one is found, then the virus replaces it with a loadable segment entry. This loadable segment will contain the virus code. The segment is set to the size of the virus, and its starting location is calculated to end just before the loadable segment with the lowest virtual address that was located earlier. The host's original entrypoint is saved in the virus code, and a new entrypoint is set to the location of the virus code in memory. The virus sets the last byte of the e_ident field to 1, as an infection marker. This has the effect of inoculating the file against a number of other viruses, since a marker in this location is quite common. Finally, the virus appends its code to the file.

The 'problem' with adding a new loadable segment to a file is that it can be seen easily in a memory map. Anyone who is familiar with the file in question will know that it has been changed.

Trimming the fat

In ordinary circumstances, the Program Header Table segment entry is redundant, since a field exists in the ELF header that points directly to it. The only missing information in the ELF header is the size of the program header table. However, this value can be calculated by using other fields from the ELF header. This is the reason why the virus uses that entry.

After all files in the directory have been examined, the virus returns control to the host.

CAVEAT EMPTOR

Along similar lines is a virus from a different author. This one was named 'Caveat' by its author and was written entirely in C – demonstrating that it can be done, though it does inject some assembly language code into the file to perform some essential operations. The virus infects files on the *Linux* platform. Despite the different authors, this virus shares many characteristics with the 'H2T3' code.

Misplaced trust

The virus begins by searching for files within the current directory. For each file that is found, the virus attempts to open and map the file. Unlike 'H2T3', if the mapping fails, this virus closes the file without attempting to unmap anything. However, this virus is equally trusting of the contents of the file. Like 'H2T3', this virus assumes that the file is in ELF format before verifying this fact. A field inside the supposed ELF header is used, without checking that the file is large enough to support its presence. A sufficiently small file will cause the code to crash. A truncated ELF file, or one with a sufficiently large value in the e_phnum field, among other things, will also cause the virus to crash, since the code contains no bounds checking of any kind.

Missing the mark

The virus is interested in ELF files which are executable, for the *Intel* x86-based CPU, and whose ABI is not specified. The virus does not check for an infection marker, because the marker is actually the absence of something instead of the presence of something. This will be explained below.

If a file is found to be infectable, then the virus rounds up the file size to a multiple of 4KB, and saves the host's original entrypoint. The rounding is required to ensure that the virus body will be completely mapped into memory later. Again, this will be explained below.

Note to self

There are two variants of the virus. Both search within the Program Header Table entries for the loadable segment that corresponds to the image base address. They also search for a PT_NOTE entry. However, the first variant ignores any PT_NOTE entry that appears before the image base address entry in the Program Header Table. This might be considered an optimization to avoid parsing the entries twice (since the entrypoint calculation requires the image base address), but some files will not be infected as a result. It could also be considered a bug, since the entrypoint calculation could be delayed until after the parsing has completed.

Force of h-ABI-t

In the case of the first variant, if an acceptable PT_NOTE entry is found, then the virus shrinks the Program Header Table by the size of one entry, to make space for the first part of the virus loader. With the PT_NOTE entry removed, the corresponding .note.ABI-tag section is unreferenced and available to be replaced. The virus overwrites the .note.ABI-tag section with the second part of the virus loader, and changes the host entrypoint to point to the first part. Since there is usually only one PT_NOTE entry in a file, its removal means that it cannot be found again. Files that do not contain a PT_NOTE entry will not be infected, obviating the need for an infection marker.

Stacking the deck

In the case of the second variant, the virus also searches for PT_PHDR and PT_GNU_STACK entries. The virus shrinks the Program Header Table by the size of these entries to make space for the entire virus loader. The virus changes the host entrypoint to point to the loader. With the removal of those entries, any subsequent examination of the file will not find sufficient space for the loader. As a result, such files will not be reinfected, obviating the need for an infection marker.

The easy way or the hard way

After the loader has been copied to the file, the virus extends the file to the multiple of 4KB that it calculated earlier, then appends the virus code. The loader works by calling the mmap() function to map into memory the virus code from the end of the file. Since the mapping requires an aligned base as a starting address, the virus must either place itself at exactly such an aligned address (the simplest case, as we see here), or the size of the mapping must be increased appropriately to potentially span two pages, and the virus code must be aware of the possibly non-zero offset within the first page where the virus body resides (which does not increase the file size to the same degree, but which increases the complexity of the algorithm and requires more code).

This method of memory-mapping the virus code avoids the loadable segment problem described above. Of course, the mapped memory might be still considered to be suspicious. The virus author described a workaround for this by allocating a new memory region and copying the virus body there before unmapping the old copy.

CONCLUSION

At first glance, the technique of replacing the .note.ABI-tag section in ELF files might appear to be similar to the .reloc overwriting technique in *Windows* PE files. However, there are far more differences than similarities, since ELF files have fewer restrictions regarding section placement, among other things. In a sense, this kind of cavity infection could be considered just another 'hole' that is being exploited.

MALWARE ANALYSIS 2

KERNEL MECHANICS OF RUSTOCK

Chandra Prakash
Sunbelt Software, USA

This article provides details of the kernel-mode operations of a recent (March 2009) version of Rustock, concentrating on the changes from its previous version. The previous version referred to in this article (Rustock.C) was detailed in an earlier issue of *Virus Bulletin* [1]. This article also describes the functions of the Rustock dropper that drops the rootkit driver.

DROPPER UNPACKING

The outer layer 1 of the dropper is packed with the well-known UPX packer. Layer 1 UPX unpacking results in a Win32 command line executable with _wmain as shown in listing 1.

```
UPX0:00401920  _wmain  proc near
UPX0:00401920  call    sub_401928
UPX0:00401925  xor     eax, eax
UPX0:00401927  retn
UPX0:00401927  _wmain        endp
UPX0:00401927
UPX0:00401928  sub_401928    proc near
UPX0:00401928  jmp     short loc_401995
        .
        .
UPX0:00401995  loc_401995:
UPX0:00401995  pusha
UPX0:00401996  or      eax, 0FFFFFFFFh
UPX0:00401999  xor     eax, 0FFFFFFFFh
UPX0:0040199C  push    2DB7h
UPX0:004019A1  push    offset loc_401957
UPX0:004019A6  retn
```

Listing 1.

The _wmain routine is a layer 2 inner custom unpacking routine which contains a lot of PUSH RETN instruction sequences. Listing 2 shows a snippet of the layer 2 unpacking routine.

```
0040197D push  offset loc_4019A7; Start of encrypted
                                ; code
00401982 pop   ebx
00401983 lea   esi, dword_420000; Starting
                                ; decryption location
00401989
00401989       loc_401989:
00401989 add   eax, 0B1788E5Ch; Decryption key
                       ; different for every dword
0040198E mov    edx, [ebx]
00401990 xor   edx, eax; Simple XOR decryption
00401992 push  edx
00401993 jmp   short loc_401941
00401941
00401941 loc_401941:
00401941 pop   dword ptr [esi]; Write decrypted dword
00401943 lea   ebx, [ebx+4]
```

Listing 2.

Listing 3 shows the start of the decrypted code after layer 2 unpacking. The decrypted code obtains the location of the process environment block (PEB) using the FS:[30] register. From PEB it retrieves the address of InitializationOrderModuleList in order to find the kernel32.dll load virtual address. This is used to resolve the import addresses of the GetProcAddress, LoadLibrary and ExitProcess APIs. These APIs are in turn used to load more libraries, e.g. advapi32.dll, and resolve functions from them.

```
00420000 8b4c2404  mov    ecx,dword ptr [esp+4]
00420004 call     0420009
00420009 pop      ebp
0042000a sub      ebp,9
0042000d mov      eax,dword ptr fs:[00000030h]
00420013 mov      eax,dword ptr [eax+0Ch]
00420016 mov      eax,dword ptr [eax+1Ch]
00420019 mov      eax,dword ptr [eax]
0042001b mov      eax,dword ptr [eax+8]; Getting
                  ;Kernel32.dll load address from PEB
```

Listing 3.

DROPPING THE ROOTKIT DRIVER

The next step is to load the rootkit driver into a shared memory. Using the CreateFileMapping API, the dropper creates a system paging file named 'shared memory section object'. The user-mode name of the section object is 'Global\5B37FB3B-984D-1E57-FF38-AA681BE5C8D9'. It then uses the virtual address return from the MapViewOfFile API to copy the rootkit driver into the shared memory.

Next, beep.sys is used as the first goat driver to install the rootkit driver. The dropper copies the beep.sys driver into a temporary file. The path to the temporary file is obtained using the GetTempPath and GetTempFileName APIs. The dropper uses the SCM APIs OpenSCManager and OpenService to get a handle to the beep service and then calls the ControlService API to stop it, if it is already running. It overwrites the beep.sys driver with its own Rustock driver and starts the beep service later using the ControlService API.

The dropper checks whether the driver has started successfully by opening a named event object created by the Rustock driver. The API used is OpenEvent and the name of the event object is 'Global\{60F9FCD0-8DD4-6453-E394-771298D2A471}'. The open event is tried several times with one-second sleep intervals until it is successful.

After the retries, the original beep.sys driver is restored from the temporary saved location. If the open event fails, the dropper uses the null.sys driver as the next goat driver, repeating the same steps. If using null.sys does not succeed either, then it creates a driver named 'glaide32.sys' in %SystemRoot%\System32\drivers and uses it to start the Rustock driver.

DRIVER AND DROPPER INTERACTION

After final unpacking in the Rustock driver, when code near the original entry point is reached, ZwOpenSection is used to open the named shared memory section object that was previously created by the dropper. In the driver, the kernel-mode section object is opened with the name '\BaseNamedObjects\5B37FB3B-984D-1E57-FF38-AA681BE5C8D9'. After opening the section object, it calls the ZwMapViewOfSection API to get the driver buffer from which to copy. The driver buffer is written to disk with a uniquely generated name that contains all hexadecimal numbers. The driver name generation is described in listing 4.

```
00010388 push edx
00010389 rdtsc
0001038B xor eax, rdtscValLoc
00010391 ror eax, 5
00010394 add eax, edx
00010396 add rdtscValLoc, eax; eax has driver name
0001039C pop edx
0001039D retn
```

Listing 4.

The driver service name in the registry is generated using the format specifier \registry\machine\system\CurrentControlSet\Services\%x to sprintf API. The driver file path is generated using the format specifier \SystemRoot\System32\drivers\%x.sys. The driver is set up in the registry as a SERVICE_SYSTEM_START service. Note, this is the same driver as beep.sys was overwritten with. The driver is written to disk by direct access to the NTFS driver, bypassing all filter drivers to evade on-access detection [1]. After writing the driver to disk and setting up the driver registry service configuration, the shared memory buffer is deallocated using ZwUnmapViewOfSection. This newly written driver will be started after the next reboot.

SIMILARITIES WITH RUSTOCK.C

The following are the similarities between this version of Rustock and the version presented in [1].

1. The decryption and decompression routines are the same at all stages, both for the Rustock driver and for the injected bot dll.
2. The number of threads started and the function of each thread remain broadly the same.
3. Both versions load private ntdll in order to obtain the SSDT index of hooked functions.
4. Both versions register a process creation notification routine using PsSetCreateProcessNotifyRoutine to search for services.exe process create events for injecting APCs.
5. Both versions create a new thread that overwrites its own driver to disk every five seconds.
6. Both versions hook the registry key parse procedure in the kernel to hide the rootkit driver service key. Normally, the parse procedure in the kernel is registered by the Configuration Manager with the Object Manager.
7. Both versions hook the ZwCreateKey, ZwOpenKey and IRP_MJ_CREATE dispatch routine of the NTFS driver.
8. Both versions send the APC1 routines to inject waitable threads in the context of services.exe.

CHANGES IN APC2

Before the APC2 routine for injecting bot dll is delivered, it communicates with the PCI bus device to get two DWORDs. One DWORD identifies the vendor and device ID of the bridge between the PCI bus to host and the other identifies the device ID of the bridge between the PCI bus and ISA bridge [2, 3]. The vendor and device IDs corresponding to these DWORD pairs are shown in Table 1 [4]. If a match occurs with any of these pairs, APC2 is not delivered [1]. Pair 1 corresponds to *VMware* and was

	Vendor ID	Device ID
Pair 1 71908086 71108086	8086 - Intel	7190 - 440BX/ZX AGPset host bridge
	8086 - Intel	7110 - PIIX4/4E/4M ISABridgeA
Pair 2 12378086 70008086	8086 - Intel	1237 - PCI & memory
	8086 - Intel	7000 - PIIX3 PCI-to-ISA bridge (Triton II)
Pair 3 71928086 71108086	8086 - Intel	7192 - 440BX/ZX chipset host-to-PCI bridge
	8086 - Intel	7110 - PIIX4/4E/4M ISBridgeA
Pair 4 11308086 1112AAAA	8086 - Intel	1130 - Host-hub interface bridge / DRAM Ctrlr
	Unknown	

Table 1: Vendor and device IDs.

checked on *VMware* versions 5.5, 6.0 and 6.5. It is more than likely that the malware uses these vendor and device IDs to detect *VMware*.

The code snippet used to obtain the device and vendor IDs corresponding to the first DWORD is shown in listing 5.

```
mov edx, 0CFBh  ; In dx set PCI mechanism control
                ; (PMC) register port number 0xCFB
in  al, dx      ; Read value from PMC register
or  al, 1       ; PCI CONFIGURATION ACCESS MECHANISM
                ; SELECT (PCAMS):
 ; Set PCI Configuration Access Mechanism #1
 ; The CONFADD and CONFDATA registers (see below)
 ; are only accessible when PCAMS = 1
out dx, al ; Enable PCI Configuration Mechanism #1

xor ebx, ebx
QRY_PCI_VENDOR_DEV_ID_LOOP:
mov eax, ebx
shl eax, 8 ; Set up device number and function number
bts eax, 1Fh ; Set bit 31. Note device no. is always 0
mov dl, 0F8h
out dx, eax   ; Output to port 0xCF8, the PCI
              ; configuration address (CONFADD) register
mov dl, 0FCh
in  eax, dx    ; Read port 0xCFC, the PCI Configuration
               ; data (CONFDATA) register
mov esi, eax
inc ax
jz  short INVALID_VALUE_READ_FR_PORT

mov eax, ebx    ; Reached here if vendID and devID are
                ; valid for the given device num and
                ; function num at bus 0
shl eax, 8
add eax, 80000008h; Set PCI Config address offset 0x8
mov dl, 0F8h
out dx, eax
mov dl, 0FCh
in  eax, dx    ; Here it is reading DWORD from CONFDATA
               ; at PCI address offset 0x8
 ; The DWORD contains four bytes as below
 ; byte at offset 0xB - broad classification
 ; byte at offset 0xA - sub-classification
 ; byte at offset 0x9 - register programming interface
 ; byte at offset 0x8 - ignored
shr eax, 8 ; Ignoring byte at offset 0x8
cmp eax, 60000h ; 0x060000 is such that
 ; 06 - PCI bridge (broad classification)
 ; 00 - bridge to CPU host (sub-classification)
 ; 00 - register programming interface
 ; Hence it identifies a PCI host bridge device

jz  short FOUND_VALID_VENDOR_DEV_ID

INVALID_VALUE_READ_FR_PORT:
inc bl  ; Here by incrementing bl which is byte sized
        ; it is scanning all device numbers and
        ; function numbers
        ; Note device number is four bits and so is
        ; the function number
jnz short QRY_PCI_VENDOR_DEV_ID_LOOP
xor esi, esi

FOUND_VALID_VENDOR_DEV_ID:
```

Listing 5.

The logic for obtaining the device and vendor ID corresponding to the second DWORD is identical except for the comparison part, as shown in listing 6.

```
cmp eax, 60100h    ; 060100 is a such that
  ; 06 - PCI bridge (broad classification)
  ; 01 - ISA bridge (sub-classification)
  ; 00 - register programming intf
  ; Hence it indentifies a PCI ISA bridge device.
jz  short loc_1116D
cmp eax, 68001h  ; 068001 identifies PCI "other"
 ; bridge device
```

Listing 6.

CREATING TCPIP HOOK

The Rustock driver creates a TCPIP hook by using the device name \Device\Tcp. First, the device object of the tcpip.sys driver is obtained by using the IoGetDeviceObjectPointer API. Next, the driver object member of the device object structure is used to get the address of the original IRP_MJ_INTERNAL_DEVICE_CONTROL dispatch routine, where the hook is placed. During the creation of this TCPIP hook it allocates two buffers of size 5,220 (0x1464) and 3,200 (0xC80) bytes from a non-paged pool. It also creates an event notification object. The purpose of these buffers and the notification object is to store data from the TCPIP hook function and then notify delivery to bot dll, as described in DispatchFunction3 and DispatchFunction4 below.

HELPER FUNCTIONS FOR TDI COMMUNICATION

Rustock uses a nice modular approach, defining a set of helper functions that are parameterized, rather than using duplicated code due to differences in parameters. Some of these helper functions relating to TDI communication are described below [6]. These helper functions are called from more than one location from various dispatch functions called to serve requests from the bot dll.

```
ReturnLockedMDLForUserBuf proc
; Remarks
; Returns MDL after locking a user-mode buffer.
; User-mode buffer is passed in requests from bot dll
; Input Parameters
; [ebp+8] - UserBufVirtAddr
; [ebp+C] - UserBufLength
; [ebp+10] - LOCK_OPERATION (IoReadAccess/
; IoWriteAccess etc.)
; Return value
; Locked MDL pointer in eax
000131A0 xor    esi, esi
000131A2 push   esi      ; Irp
000131A3 push   esi      ; ChargeQuota
000131A4 push   esi      ; SecondaryBuffer
000131A5 push   dword ptr [ebp+0Ch]  ; UserBufLength
000131A8 push   dword ptr [ebp+8]  ; UserBufVirtAddr
000131AB call   ds:IoAllocateMdl
000131B1 mov    edi, eax ; Store MDL pointer in edi
 .
 .
000131BD push   dword ptr [ebp+10h]  ; LOCK_OPERATION
000131C0 push   1        ; AccessMode = UserMode
000131C2 push   edi      ; MDL
000131C3 call   ds:MmProbeAndLockPages ; Lock
                                ;user-mode buffer
 .
 .
000131E1 mov    eax, edi ; Return MDL pointer in eax
ReturnLockedMDLForUserBuf endp
```

Listing 7.

```
PrepareAndSendIrp proc
; Remarks
; In regard to the bot dll requests, this function
; prepares TDI IRPs for the TCPIP.sys driver. It fills
; in MDL, Next Stack Location parameters and then
; sends it to the TCP driver. IRP completion and post
; processing cleanup is also handled.
; Input parameters
; eax - control structure such that:
;       [eax+20h] = allocated KEVENT object
;       [eax+30h] = reusable IRP pointer
;       [eax+38h] = placeholder for output
;       IRP->IoStatus.Information
;
; Return value
; NTSTATUS in eax copied from local var 'status'
;
; Local vars
; [ebp-8] - PMDL MemoryDescriptorList
; [ebp-4] - NTSTATUS status
;
00012D1C mov  esi, eax
00012D1E mov  edi, [esi+30h] ; Get IRP
.
.
00012D28 lea  ebx, [esi+20h]
00012D2B push ebx        ; KEVENT object
00012D2C mov  [ebp+MemoryDescriptorList], eax
00012D2F call ds:KeInitializeEvent
00012D35 mov  eax, [esi+30h]     ; Get IRP pointer
00012D38 mov  eax, [eax+60h]
 ; IRP->Tail.Overlay.CurrentStackLocation
00012D3B sub  eax, 24h   ; IoGetNextIrpStackLocation
00012D3E mov  dword ptr [eax+1Ch], offset
TcpIrpCompleteionRoutine;
 ; Set IO_STACK_LOCATION.CompletionRoutine
00012D45 mov  [eax+20h], ebx ; Set IRP Event object
 ; in IO_STACK_LOCATION.Context
00012D48 mov  byte ptr [eax+3], 0E0h ; Set
 ; IO_STACK_LOCATION.Control
00012D4C mov  ecx, [esi+14h]    ; TCP DeviceObject
00012D4F mov  edx, edi          ; IRP
00012D51 call ds:IofCallDriver ; Call TCPIP driver
.
.
00012D86 mov  eax, [edi+1Ch]
; Irp->IoStatus.Information
00012D89 mov  [esi+38h], eax
00012D8C mov  eax, [edi+18h] ; Irp->IoStatus.Status
00012D8F mov  [ebp+status], eax
.
.
00012D92 mov  esi, [ebp+MemoryDescriptorList]
.
.
00012D99 test byte ptr [esi+6], 2 ; Compare
 ; MDL_PAGES_LOCKED flag
00012D9D jz    short LOC_DONT_UNLOCKPAGES
00012D9F push esi  ; MemoryDescriptorList
00012DA0 call ds:MmUnlockPages; Unlock pages
00012DA6 LOC_DONT_UNLOCKPAGES:
00012DA6 push esi  ; Mdl
00012DA7 call ds:IoFreeMdl
.
.
00012DAF push edi  ; Don't free IRP. Reuse it!
00012DB0 call ds:IoReuseIrp
00012DB6 mov  eax, [ebp+status]; Return
; NTSTATUS in eax
.
.
PrepareAndSendIrp endp
```

Listing 8.

ZwCreateEvent HOOK

In contrast to the previous version of Rustock, which hooked ZwTerminateProcess, this one hooks ZwCreateEvent for communication between the user-mode bot dll and the driver [1].

```
NTSTATUS
   ZwCreateEvent(
      OUT PHANDLE  EventHandle,
      IN ACCESS_MASK  DesiredAccess,
      IN POBJECT_ATTRIBUTES  ObjectAttributes,
      IN EVENT_TYPE  EventType,
      IN BOOLEAN  InitialState
      );
```

In the ZwCreateEvent API the DesiredAccess parameter value 0x0DC17E241 is used to delineate messages from bot dll versus other normal calls to this API. If ZwCreateEvent was invoked from bot dll, as indicated by the DesiredAccess value, then EventHandle contains a structure describing the input/output parameters. The layout of this structure is shown in listing 9.

```
ZwCreateEventHook proc
; Remarks
; This hook is used for Rustock bot dll and
; driver communication. The communication structure
; (RustockBotDllDrvComm) layout is:
; +0x0 FunctionIndex     // Index into function array
; +0x4 InputBuffer       // Input buffer
; +0x8 InputBufferSize   // Input buffer size
; +0xC OutputBuffer      // Output buffer
; +0x10 OutputBufferSize // Output buffer size
;
; Input parameters
; [ebp+8] - OUT PHANDLE EventHandle
; [ebp+C] - IN ACCESS_MASK DesiredAccess
; [ebp+10] - IN POBJECT_ATTRIBUTES ObjectAttributes
; [ebp+14] - IN EVENT_TYPE EventType
; [ebp+18] - IN BOOLEAN InitialState
;
.
.
00012849 cmp [ebp+DesiredAccess], 0DC17E241h
00012850 jnz short CALL_ORIGINAL_ZwCreateEvent
; When bot dll encoded value doesn't match, call
; original ZwCreateEvent function
00012852 call ds:IoGetCurrentProcess
00012858 cmp eax, [ServicesEPROCESSValue]
0001285E jnz short CALL_ORIGINAL_ZwCreateEvent
; If not services.exe, handle requests through
; original ZwCreateEvent function
.
.
00012864 push 1 ; Alignment
00012866 push 14h ; Size of RustockBotDllDrvComm
00012868 mov esi, [ebp+EventHandle]
0001286B push esi ; Pointer to RustockBotDllDrvComm
0001286C mov edi, ds:ProbeForRead
00012872 call edi ; Check if structure is readable
00012874 push 1 ; Alignment
00012876 push 14h ; Size of RustockBotDllDrvComm
00012878 push esi ; Pointer to RustockBotDllDrvComm
00012879 mov ebx, ds:ProbeForWrite
0001287F call ebx ; Check if structure is writable
00012881 cmp dword ptr [esi], 0Ch ; Check function
; index bound
00012884 jnb short LOC_SET_ERROR_RETURN ; Only 11
; functions are registered. More validation
; on InputBuffer and OutputBuffer done next
.
000128A4 mov eax, [esi] ; eax has FunctionIndex
000128A6 push esi ; address of
; RustockBotDllDrvComm
000128A7 call ZwCrtEvtDispFuncsArr[eax*4] ; Call
; dispatch function based on FunctionIndex
.
.
000128CD CALL_ORIGINAL_ZwCreateEvent:
000128CD push [ebp+InitialState]
000128D0 push [ebp+EventType]
000128D3 push [ebp+ObjectAttributes]
000128D6 push [ebp+DesiredAccess]
000128D9 push [ebp+EventHandle]
000128DC mov eax, OrigZwCreateEventAdr
000128E1 call dword ptr [eax]
.
.
LOC_SET_ERROR_RETURN:
.
.
ZwCreateEventHook endp
```

Listing 9.

DISPATCH FUNCTIONS IN ZwCreateEvent HOOK

There are 11 dispatch functions that can be called from ZwCreateEvent hook. These are used for communication between bot dll and the driver and their general details are described below. Any disk I/Os in these dispatch functions are done by direct access to the NTFS driver.

DispatchFunction1

This function is used to overwrite the existing Rustock driver with a new one sent from the bot dll. The new driver buffer and its size are passed in the InputBuffer and InputBufferSize parameters of the RustockBotDllDrvComm structure.

DispatchFunction2

This dispatch function deletes its own driver from the disk.

DispatchFunction3

This function is used for copying to user-mode data that was filtered in the Rustock TCPIP hook in TCP_SEND. When the data is ready, the TCPIP hook calls KeSetEvent and this function waits on that event (see listings 10 and 16).

```
DispatchFunction3 proc
.
00012954 push   ebx   ; Waitable Event Object
00012955 call   ds:KeWaitForSingleObject ; Wait
 ; for notication from TCP hook
 .
 .
00012975 mov    ecx, ebx        ; FastMutex
00012977 call   ds:ExAcquireFastMutex ; Get copy lock
 ; to sync with TCP hook before copy
0001297D mov    eax, BufferAddrToCopyFrom
00012982 mov    ecx, ebp ; Set up buffer length in ecx
 .
 .
0001298A mov    esi, eax ; Set up source buffer
0001298C mov    [edi+OutputBufferSize], ecx ; Set
 ; output buffer size
```

```
0001298F mov    edi, [edi+OutputBuffer] ; Set
 ; output/destination buffer
00012992 mov    edx, ecx
00012994 shr    ecx, 2
00012997 rep movsd  ; Copy in chunks of dwords
00012999 mov    ecx, edx
0001299B and    ecx, 3
0001299E rep movsb ; Copy remainder in chunk of bytes
.
.
000129AD call   ds:ExReleaseFastMutex; Copy done,
 ; release copy lock
.
DispatchFunction3 endp
```

Listing 10.

DispatchFunction4

This function returns to bot dll the process name captured in the TCPIP hook, which sent the data in a certain format through TDI_SEND (see listing 16).

DispatchFunction5

This function is called on request from bot dll for creating a new unique TCPIP connection control structure. This control structure is an array of 16 DWORDs that contains, for example, the address of reusable IRPs (see listings 8, 12 and 13), TDI connection context handles, TDI file objects etc. There is an array of 10,000 such control structures. When bot dll requests the driver through dispatch functions relating to TCP activity, the bot dll sends in its input parameter an index into the array of control structures. Listing 11 shows a common routine used to obtain the address of the control structure using the array index.

```
GetCtrlStructByIndex proc
; Remarks
; This function returns address of TCPIP control
; structure from an array of these structures
; Input parameter
; eax - Index into the array of control structures
000133CE cmp  eax, 2710h ; compare to 10000
000133D3 jle   short LOC_VALID_ARRAY_INDEX
000133D5 xor   eax, eax ; return NULL
000133D7 retn
000133D8 LOC_VALID_ARRAY_INDEX:
000133D8 mov  ecx, MasterArrCtrlStructs
000133DE mov eax, [ecx+eax*4] ; return pointer to
; control structure by array index
000133E1 retn
GetCtrlStructByIndex endp
```

Listing 11.

DispatchFunction6

This is used to clean up a control structure allocated in DispatchFunction5. The index of the control structure to deallocate is passed in the InputBuffer. The cleanup includes sending TDI disconnect, closing TDI transport address, TDI connection context, freeing IRP and deallocating the control structure (see listing 12).

```
DispatchFunction6 proc
.
00013459 mov  eax, esi   ; Set index of control struct
; in eax
0001345B call GetCtrlStructByIndex
00013460 test eax, eax
00013462 jz   short loc_13475
00013464 push eax ; Pass address of control struct
00013465 call SendDisconnAndDeAllocCtrlStruct
.
DispatchFunction6 endp

SendDisconnAndDeAllocCtrlStruct proc
; Input parameter
; [ebp+8] - Control structure
.
00012DF6 mov     esi, [ebp+8]
00012E23 mov     eax, [esi+30h]; IRP
00012E26 mov     eax, [eax+60h]
 ; IRP->Tail.Overaly.CurrentStackLocation
00012E29 sub     eax, 24h
 ; IoGetNextIrpStackLocation
00012E2C mov     byte ptr [eax], 0Fh
 ; IRP_MJ_INTERNAL_DEVICE_CONTROL
00012E2F mov     byte ptr [eax+1], 6
; TDI_DISCONNECT
.
00012E52 mov     eax, esi
00012E54 call    PrepareAndSendIrp ; Send
; disconnect remaining cleanup done next
.
SendDisconnAndDeAllocCtrlStruct endp
```

Listing 12.

DispatchFunction7

This function is used to connect to a remote host. The InputBuffer contains an array index of the control structure and IP address and port to which to connect. Figure 1 shows a portion of netstat output showing Rustock connected to the SMTP port on several remote hosts.

```
DispatchFunction7 proc
.
00013066 mov     ecx, [eax+30h]; IRP
00013069 mov     ecx, [ecx+60h]
 ; IRP->Tail.Overaly.CurrentStackLocation
0001306C sub     ecx, 24h
 ; IoGetNextIrpStackLocation
0001306F push    esi
00013070 mov     byte ptr [ecx], 0Fh
 ; IRP_MJ_INTERNAL_DEVICE_CONTROL
00013073 mov     byte ptr [ecx+1], 1
 ; TDI_ASSOCIATE_ADDRESS
00013077 mov     esi, [eax+14h] ; Get TCP device
 ; object from control struct
0001307A mov     [ecx+14h], esi
```

```
; Set IO_STACK_LOCATION.DeviceObject
0001307D mov    esi, [eax+0Ch] ; Get TDI
; ConnContext
; FileObject from control struct
00013080 mov    [ecx+18h], esi ; Set
; ConnContext in
; IO_STACK_LOCATION.FileObject
.
0001308D call   PrepareAndSendIrp; Send IRP
.
00013154 sub    eax, 24h
00013157 mov    byte ptr [eax], 0Fh
; IRP_MJ_INTERNAL_DEV_CTRL
0001315A mov    byte ptr [eax+1], 3 ;
TDI_CONNECT
.
0001316A lea    ecx, [ebp+tdiConnInfo]
0001316D mov    [eax+8], ecx
; TDI_REQUEST.RequestConnectionInfo
00013170 lea   ecx, [ebp+tdiConnInfo]
00013173 mov   [eax+0Ch], ecx
; TDI_REQUEST.ReturnConnectionInfo
00013176 mov   [eax+10h], ebx
; TDI_REQUEST.RquestSpecific=0 (ebx)
.
0001317C call PrepAndSendIrp; Send IRP
.
DispatchFunction7 endp
```

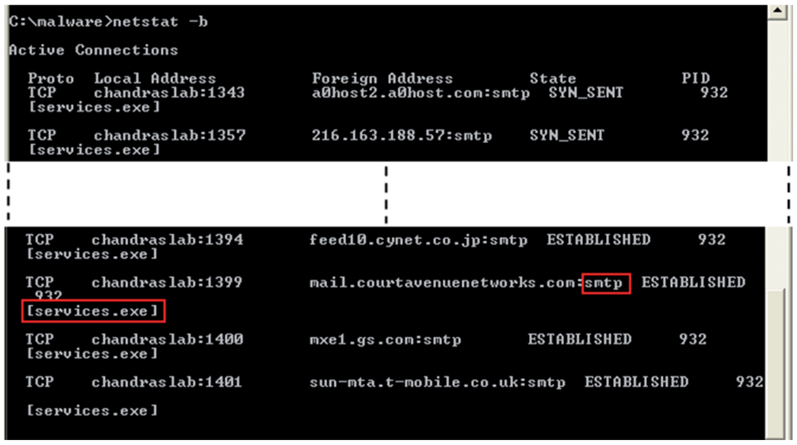
Listing 13.

DispatchFunction8

This function is used to send data using TDI_SEND. The index of the control structure and data to send is passed in the InputBuffer.

```
DispatchFunction8 proc
.
00013206 xor ebx, ebx
00013208 push ebx
; 0 - IoOperationRead
00013209 push edi
0001320A push [ebp+arg_4]
0001320D call ReturnLockedMDLForUserBuf; Get
; Locked MDL for read
00013212 cmp eax, ebx
00013214 jz short ErrorExit
.
00013228 mov ecx, [esi+30h]
0001322B mov ecx, [ecx+60h]

0001322E sub ecx, 24h; IoGetNextIrpStackLocation
00013231 mov byte ptr [ecx], 0Fh
; IRP_MJ_INTERNAL_DEVICE_CONTROL
00013234 mov byte ptr [ecx+1], 7 ; TDI_SEND
.
; From Control Struct get TCP DeviceObject and TDI
; ConnContext FileObject to fill in corresponding
; fields in IO_STACK_LOCATION
.
; Set TDI_REQUEST_KERNEL_SEND.SendFlags
00013247 mov [ecx+4], edi
```



```
C:\malware>netstat -b

Active Connections

  Proto  Local Address          Foreign Address        State           PID
  TCP    chandraslab:1343       a0host2.a0host.com:smtp  SYN_SENT        932
  [services.exe]

  TCP    chandraslab:1357       216.163.188.57:smtp    SYN_SENT        932
  [services.exe]

  TCP    chandraslab:1394       feed10.cynet.co.jp:smtp  ESTABLISHED     932
  [services.exe]

  TCP    chandraslab:1399       mail.courtavenuenetworks.com:smtp  ESTABLISHED
   932
  [services.exe]

  TCP    chandraslab:1400       mxe1.gs.com:smtp       ESTABLISHED     932
  [services.exe]

  TCP    chandraslab:1401       sun-mta.t-mobile.co.uk:smtp  ESTABLISHED     932

  [services.exe]
```

Figure 1: netstat output showing Rustock connected to the SMTP port on several remote hosts.

```
; Set TDI_REQUEST_KERNEL_SEND.SendLength
0001324A mov ecx, [esi+30h]
; Get IRP from Control Struct
0001324D mov [ecx+4], eax ; Set IRP->MdlAddress
00013250 push dword ptr [esi+3Ch]
00013253 mov eax, esi
00013255 call PrepareAndSendIrp
.
DispatchFunction8 endp
```

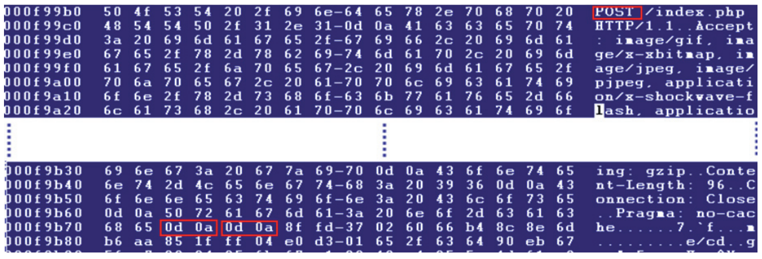
Listing 14.

Figure 2 shows a screenshot of data sent through DispatchFunction8.

DispatchFunction9

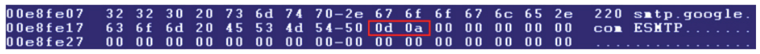
This function is used to receive data using TDI_RECEIVE. The index of the control structure is passed in the InputBuffer and the address of the receive buffer is passed in the OutputBuffer.

```
DispatchFunction9 proc
.
0001328B push 1 ; 1 - IoOperationWrite
0001328D push edi
0001328E push [ebp+buffer]
00013291 call ReturnLockedMDLForUserBuf ; Get Locked
; MDL for write
00013296 test eax, eax
00013298 jz short ErrorExit
.
000132AF mov ecx, [esi+30h]
000132B2 mov ecx, [ecx+60h]
000132B5 sub ecx, 24h; ; IoGetNextIrpStackLocation
000132B8 mov byte ptr [ecx], 0Fh
; IRP_MJ_INTERNAL_DEVICE_CONTROL
000132BB mov byte ptr [ecx+1], 8 ; TDI_RECEIVE
.
; From Control Struct get TCP DeviceObject and TDI
; ConnContext FileObject to fill in corresponding
; fields in IO_STACK_LOCATION
```



```
000f99b0  50 4f 53 54 20 2f 69 6e-64 65 78 2e 70 68 70 20  POST /index.php
000f99c0  48 54 54 50 2f 31 2e 31-0d 0a 41 63 63 65 70 74  HTTP/1.1..Accept
000f99d0  3a 20 69 6d 61 67 65 2f-67 69 66 2c 20 69 6d 61  : image/gif, ima
000f99e0  67 65 2f 78 2d 78 62 69-74 6d 61 70 2c 20 69 6d  ge/x-xbitmap, im
000f99f0  61 67 65 2f 6a 70 65 67-2c 20 69 6d 61 67 65 2f  age/jpeg, image/
000f9a00  70 6a 70 65 67 2c 20 61-70 70 6c 69 63 61 74 69  pjpeg, applicati
000f9a10  6f 6e 2f 78 2d 73 68 6f-63 6b 77 61 76 65 2d 66  on/x-shockwave-f
000f9a20  6c 61 73 68 2c 20 61 70-70 6c 69 63 61 74 69 6f  lash, applicatio
   :                                  :                                   :
000f9b30  69 6e 67 3a 20 67 7a 69-70 0d 0a 43 6f 6e 74 65  ing: gzip..Conte
000f9b40  6e 74 2d 4c 65 6e 67 74-68 3a 20 39 36 0d 0a 43  nt-Length: 96..C
000f9b50  6f 6e 6e 65 63 74 69 6f-6e 3a 20 43 6c 6f 73 65  onnection: Close
000f9b60  0d 0a 50 72 61 67 6d 61-3a 20 6e 6f 2d 63 61 63  ..Pragma: no-cac
000f9b70  68 65 0d 0a 0d 0a 8f fd-37 02 60 66 b4 8c 8e 6d  he......7.`f...m
000f9b80  b6 aa 85 1f ff 04 e0 d3-01 65 2f 63 64 90 eb 67  .........e/cd..g
```

Figure 2: Screenshot of data sent through DispatchFunction8.



```
00e8fe07  32 32 30 20 73 6d 74 70-2e 67 6f 6f 67 6c 65 2e  220 smtp.google.
00e8fe17  63 6f 6d 20 45 53 4d 54-50 0d 0a 00 00 00 00 00  com ESMTP.......
00e8fe27  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
```

Figure 3: Memory dump of receive buffer after a series of single-byte receives.

```
.
000132CB mov dword ptr [ecx+8], 20h
; TDI_REQUEST_KERNEL_RECEIVE.ReceiveFlags=TDI_
RECEIVE_NORMAL
000132D2 mov [ecx+4], edi
; TDI_REQUEST_KERNEL_RECEIVE.ReceiveLength
000132D5 mov ecx, [esi+30h] ; ; Get IRP from Control
; Struct
000132D8 mov [ecx+4], eax ; Set IRP->MdlAddress
000132DB push dword ptr [esi+3Ch]
000132DE mov eax, esi
000132E0 call PrepareAndSendIrp
.
DispatchFunction9 endp
```

Listing 15.

Figure 3 shows the memory dump of the receive buffer after a series of single-byte receives. Note that every data-sent packet sent or received by Rustock is delimited by CRLF (0xd 0xa) by sequence.

DispatchFunctions 10 and 11 return some state information upon request from bot dll.

TCPIP HOOK

The TCPIP hook is mainly on two TDI requests: TDI_SEND and TDI_CONNECT (see listing 16). The hook on TDI_SEND checks for the ‘RCPT TO:’ ASCII string at the beginning of sent data. If a match occurs, the string between angular brackets (left anchor ‘<’ and right anchor ‘>’) is extracted and copied to a memory buffer and then an event is signalled for DispatchFunction3. Also, after finding the matching data this hook obtains the process name of the process sending the data and stores it in a memory buffer for DispatchFunction4. This ‘RCPT TO:’ string field appears to correspond to the recipient’s email address in the SMTP protocol [5].

The part of the hook relating to TDI_CONNECT monitors the number of connection attempts to SMTP port 25 on a per-process basis [5]. The EPROCESS of the process requesting connect is stored in an array of maximum size 400. Every time the process makes an attempt to connect the connect count is incremented using EPROCESS as the key. If the connect count exceeds 200, the connect IRP is completed right away with NT status STATUS_UNSUCCESSFUL (0xC0000001), which returns failure to the user-mode application attempting to connect.

```
         HookTCPDevCtrlDispFunc proc
         ; Input Parameter
         ; [esp+8] - DeviceObject
         ; [esp+C] - Irp
         .
00012A91 mov ebx, [esp+Irp]
00012A95 cmp byte ptr [ebx+20h], 0 ; Check
; Irp.RequestorMode is Kernel
.
00012A9B mov edi, [ebx+60h]
; Irp->Tail.Overlay.GetCurrentStackLocation
00012A9E mov [esp+8+Irp], edi
00012AA2 jz CheckTDI_CONNECT; Jump if RequestorMode
; is Kernel
00012AA8 cmp byte ptr [edi+1], 7; TDI_SEND
00012AAC jnz CheckTDI_CONNECT; Jump if MinorFunction
; != TDI_SEND
.
; If MinorFunction==TDI_SEND and TCP send buffer range
; is >= 10 and < 100 bytes, then check for “RCPT TO:”
; at the buffer start (esi) as below
00012AE8 cmp dword ptr [esi], 54504352h;
; Compare buffer to “RCPT”
00012AEE jnz CheckTDI_CONNECT
00012AF4 cmp dword ptr [esi+4], 3A4F5420h
; Compare buffer+4 to “ TO:”
.
00012B17 add esi, 8; Increment esi past “RCPT TO:”
; In the remaining send buffer, extract an ASCII
; string between ‘<’ and ‘>’ delimiters
.
00012BA7 push offset SomeNotfEvtObj1 ; Event
00012BAC call ds:KeSetEvent ; If matching string
; found, set event for DispatchFunction3
.
CheckTDI_CONNECT:
00012BBD cmp byte ptr [edi+1], 3 ; TDI_CONNECT
00012BC1 jnz short CallTCPOrigDevCtrlFunc ; Jump if
; not TDI_CONNECT
.
00012BCF cmp word ptr [eax+6], 2
; TA_ADDRESS.AddressType == TDI_ADDRESS_TYPE_IP
00012BD4 jnz short CallTCPOrigDevCtrlFunc
00012BD6 cmp byte ptr [ebx+20h], 0
; Check Irp.RequestorMode
00012BDA jz short CallTCPOrigDevCtrlFunc ; Jump if
; Irp.RequestorMode == Kernel
00012BDC cmp word ptr [eax+8], 1900h
```

```
00012BE2 jnz short CallTCPOrigDevCtrlFunc ; Jump if
; port != 25 (0x19)
.
00012BED call CheckPortConnAttmptsFrThisProc; This
; function returns TRUE if max number of connection
; attempts for the current process is >= 200.
00012BF2 test eax, eax
00012BF4 jz short CallTCPOrigDevCtrlFunc; If
; function returned NON-ZERO complete request with
; STATUS_UNSUCCESSFUL
00012BF6 and dword ptr [ebx+1Ch], 0
; Irp.IoStatus.Information = 0
00012BFA mov esi, 0C0000001h ; STATUS_UNSUCCESSFUL
00012BFF xor dl, dl ; PriorityBoost
00012C01 mov ecx, ebx ; IRP
00012C03 mov [ebx+18h], esi
; IRP.IoStatus.Status=STATUS_UNSUCCESSFUL
00012C06 call ds:IofCompleteRequest ; Request is
; completed here and NOT sent to lower driver.
00012C0C mov eax, esi
00012C0E jmp short FunctionExit
.
00012C10 CallTCPOrigDevCtrlFunc:
.
00012C10 push ebx
00012C11 push [esp+0Ch+DeviceObject]
00012C15 call OrigTCPDevCtrlDispFunc
.
FunctionExit:
.
HookTCPDevCtrlDispFunc endp
```

Listing 16.

REFERENCES

[1] Prakash, C. Your filters are bypassed: Rustock.C in the kernel. Virus Bulletin, November 2008, p.6. http://www.virusbtn.com/pdf/magazine/2008/200811.pdf.

[2] 82434LX/82434NX PCI, CACHE AND MEMORYCONTROLLER (PCMC). http://datasheet.digchip.com/227/227-3-008971-2434NX.pdf.

[3] PCI 2.2 local bus specification. http://www.ece.mtu.edu/faculty/btdavis/courses/mtu_ee3173_f04/papers/PCI_22.pdf.

[4] PCI vendor and device lists. http://www.pcidatabase.com/.

[5] Simple Mail Transfer Protocol. http://en.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol.

[6] TDI drivers. http://msdn.microsoft.com/en-us/library/aa505007.aspx.

FEATURE 1

EARLY WARNING APPROACHES TO COMBAT TYPOSQUATTING

Amit Verma
Symantec, India

Typosquatting is a practice that takes advantage of the typographical mistakes (typos) often made by users when entering a website address into a web browser. For example, 'virustbn.com' is a typo of 'virusbtn.com' in which the letters 'b' and 't' are interchanged [*very easily done! - Ed*].

The low cost of domain registration makes it inexpensive for malicious users to register multiple domains that are very similar to those of popular brands, but which incorporate the sort of typographical error that might be made by a user when entering the URL. These domains are then used to host advertisements and pornographic material, to distribute malware, or are set up for phishing.

A similar practice of email typosquatting has also emerged, which involves registering common misspellings of domain names and setting up mail servers on them to listen for client connections. If a user happens to misspell the domain name while typing an email address, the mail is sent unintentionally to a nefarious recipient. This can lead to data loss from organizations.

Typosquatting can also prove harmful for important events such as the US presidential elections wherein candidates raise funds for election campaigns through their websites. Candidates' websites often have domain names of considerable character length. If a contributor mistypes the website address – which is not unlikely considering its length and given that the domain name may not have been encountered previously – he might be taken to a site controlled by a typosquatter from which his personal and financial details may be stolen. Furthermore, an increasing number of children are spending time on the Internet, and they are particularly susceptible to this kind of attack.

There is a need for effective techniques to combat this Internet-borne threat. Auto-completion in browsers and email clients mitigates the risk to some extent (recognizing the initial characters typed by the user and automatically filling in the rest, thus reducing the risk of the user mistyping the URL), but it is ineffective for URLs which have not previously been encountered by the browser. The effectiveness of URL reputation-based techniques is also limited given the highly dynamic nature of the content posted on typosquatted domains.

In this article, a two-step approach is discussed which prioritizes the registration of domain typos and effectively detects typos entered into Internet browsers and email clients. The rest of the article is organized as follows: the next section discusses a probabilistic typographical analysis technique that determines the relative frequency of occurrence of typos based on users' typing error patterns. This is followed by a discussion of a proposed metric named Probabilistic String Similarity Index (PSSI), which is a modified form of the above approach to efficiently detect typos at browsers, email clients and the like. The article closes with a description of the specific areas in which such techniques could usefully be applied.

PROBABILISTIC TYPOGRAPHICAL ANALYSIS

The first step towards preventing typosquatting should be taken by the entity registering a new domain. Ideally, in addition to registering the new domain, it should register as many typos and look-alikes of the original domain as possible so that even if users mistype the domain name, they will be redirected to the intended website. However, a typical five- to seven-character domain name has nearly 400 potential typos – and even large organizations would struggle to register all of those as domains. As the number of characters in the domain name increases, the number of possible typos grows exponentially. Also, some typo domains may, intentionally or unintentionally, already have been registered by others.

What is needed is a way to identify the typos that have a higher likelihood of occurring compared to others so that the registering of 'unwanted' domain names can be prioritized. This can be done using a schema to rank typos based on their frequency of occurrence.

The following are the kinds of typographical error that are usually exploited by typosquatters:

1. *Skip letter*: a character is skipped. For example: www.vrusbtn.com.
2. *Double letter*: a character is typed twice. For example: www.virussbtn.com.
3. *Reverse letter*: two successive characters are interchanged. For example: www.virustbn.com.
4. *Missed key*: a character is mistyped with an adjacent key on the keyboard. For example: www.viruabtn.com – here 'a' is typed instead of 's'.
5. *Inserted key*: an extra character is inserted while typing. For example: www.virusdbtn.com.
6. *Missing dot*: sometimes the dot is omitted as in http://wwwvirusbtn.com.

7. Any combination of the above: multiple typographical errors can occur, although the odds of this are quite a lot lower.

We propose a technique of ranking typos based on the probabilistic analysis of typed data. An estimate of the probability of each of the typographical errors mentioned above can be determined by an algorithmic analysis of data gathered from typing test tools. Alternatively, data can be collected from Internet users by the use of Human-Based Computation (HBC) games. These probabilities can be used further to calculate the rank of a domain typo as given below.

Using error probabilities to calculate rank of a typo

The following probabilities are collected from an analysis of typed data:

– Pr(Skip letter) - Ps(a), ….Ps (z), Ps (a | x)….Ps (f | s)

where Ps(a) is the probability of skipping character 'a' and Ps(a|x) is the probability of skipping 'a' given that 'x' was the previous character.

– Pr(Double letter) - Pd(a), ….Pd (z), Pd (a | x)….Pd (f | s)

where Pd(a) is the probability of entering 'a' twice and Ps(a|x) is the probability of entering 'a' twice given that 'x' was the previous character.

– Pr(Reverse letter) - Pr(a | x), ….Pr (x | a)…..

where Pr(a|x) is the probability of interchanging 'a' with 'x', etc.

– Pr(Missed) - Pm(w | s), Pm (d | s), Pm (a | s), Pm (x | s), ...

where Pm(w|s) is the probability of missing 'w' given that 's' was the previous character.

– Pr(Skip | Missed) is the probability of a 'Skip' error given that a 'Missed' error has occurred.

For example, let us consider the domain name 'virusbtn'. Using the probabilities above, the rank of the typo 'vrustbn', in which character 'i' is missed, and 'b' and 't' are interchanged, is calculated as:

– Rank(vrustbn) = Pr(Skip) * Ps(i) * Pr(Missed | Skip) * Pm(b | t) * C

where C is a normalization factor.

Figure 1 shows a block diagram for a method that generates and ranks the typos for a given domain name using the probabilistic ranking technique.

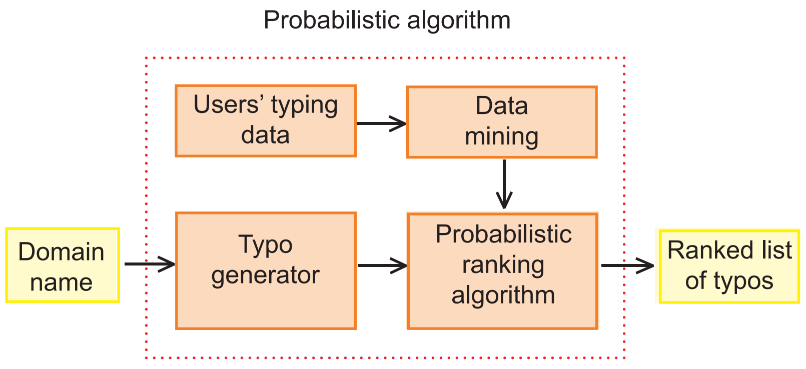


Figure 1: Method for probabilistic ranking of typographical errors.

An approximate probabilistic model was formulated with this approach to rank typos. Figure 2 shows the results for the domain www.virusbtn.com.

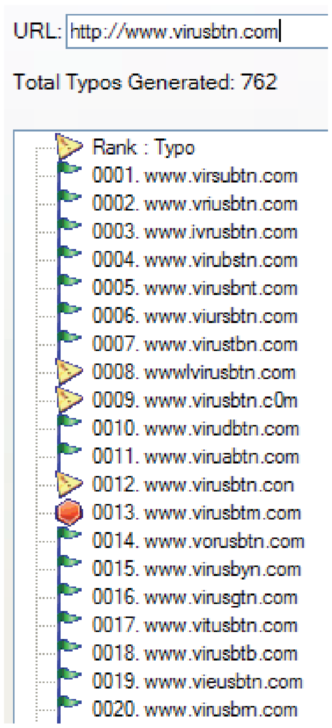
Figure 2 (a) shows the top 20 typo domains (out of 762) for www.virusbtn.com. The domain numbered 13 in Figure 2 (a) existed at the time of conducting this experiment. Figure 2 (b) shows the overall probability values of the top typo domains. Note that these results do not take into account additional information such as the top-level domain – for example,'c0m' (character 'o' replaced with zero) will not be a possible typo. The demarcation between the overall probabilities of typos can be made finer by using a more accurate probabilistic model.

Using the model, institutions whose sites may be prone to typosquatting can obtain a prioritized list of frequently occurring unwanted domains. They can then register those domains or try to buy them from others. Alternatively, typo domains which cannot be registered (for any reason) can be monitored using visual similarity methods for content that matches that of the legitimate website (which may be a sign of a phishing site). For optimized monitoring, the relative probabilities of the typos can be used to determine the frequency with which the typo domains should be monitored.

TYPO DETECTION AT BROWSERS

Another step towards preventing typosquatting is to monitor the URLs typed by users into their browsers and email clients.

The goal here is to be able to effectively detect common typos at the browser, web proxy or Internet gateway. To accomplish this, a user-specific whitelist can be constructed using email client and browser histories, favourite links,

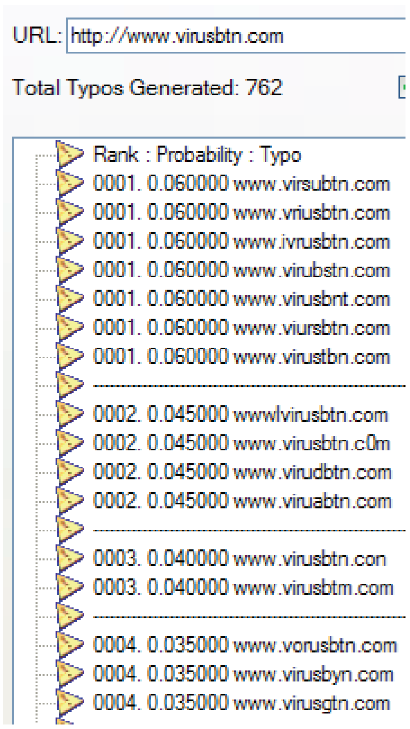


URL: http://www.virusbtn.com

Total Typos Generated: 762

```
Rank : Typo
0001. www.virsubtn.com
0002. www.vriusbtn.com
0003. www.ivrusbtn.com
0004. www.virubstn.com
0005. www.virusbnt.com
0006. www.viursbtn.com
0007. www.virustbn.com
0008. wwwlvirusbtn.com
0009. www.virusbtn.c0m
0010. www.virudbtn.com
0011. www.viruabtn.com
0012. www.virusbtn.con
0013. www.virusbtm.com
0014. www.vorusbtn.com
0015. www.virusbyn.com
0016. www.virusgtn.com
0017. www.vitusbtn.com
0018. www.virusbtb.com
0019. www.vieusbtn.com
0020. www.virusbm.com
```

(a)



URL: http://www.virusbtn.com

Total Typos Generated: 762

```
Rank : Probability : Typo
0001. 0.060000 www.virsubtn.com
0001. 0.060000 www.vriusbtn.com
0001. 0.060000 www.ivrusbtn.com
0001. 0.060000 www.virubstn.com
0001. 0.060000 www.virusbnt.com
0001. 0.060000 www.viursbtn.com
0001. 0.060000 www.virustbn.com
-----------------------------------
0002. 0.045000 wwwlvirusbtn.com
0002. 0.045000 www.virusbtn.c0m
0002. 0.045000 www.virudbtn.com
0002. 0.045000 www.viruabtn.com
-----------------------------------
0003. 0.040000 www.virusbtn.con
0003. 0.040000 www.virusbtm.com
-----------------------------------
0004. 0.035000 www.vorusbtn.com
0004. 0.035000 www.virusbyn.com
0004. 0.035000 www.virusgtn.com
```

(b)

Figure 2 (a): ranked list of typos for www.virusbtn.com; (b): ranked typo list along with probabilities.

safe site lists and the like. This can be supplemented with lists of popular banking/trading/commerce websites that can be provided by security vendors. A possible approach could be to generate typos for each entry in this whitelist and add the top ranked typo domains for each to the restricted site list of a browser. If a user makes a typographical mistake in typing any domain in the whitelist, he can be given an appropriate warning message. Given the relative ranks of various typo domains, various policies can be introduced. For example, all typos up to rank 10 can be blocked; warnings can be given for typos up to rank 50; and typos ranked greater than 50 can be allowed since they are less likely to be typosquatted domains. Another policy could be to block all typos up to rank 20 (strict monitoring) if the user has a frequent record of visiting the corresponding URL(s).

The probabilistic approach to find relative probabilities can also be modified to cater to typo detection at the users' end. To eliminate the need for storing possible typos, a string similarity metric is proposed which takes into account both the string Hamming distance and the probabilities of typographical errors. This metric eliminates the need to keep a list of typos of popular URLs at the browser and thus typo detection can be carried out more efficiently. The following is the equation of such a metric called Probabilistic String Similarity Index (PSSI):

$$PSSI(S_1, S_2) = \sum_{\substack{\forall String \\ Alignments}} \left(HD \sum_{i=1}^{M} HD_{\Delta} \log_e \left(\frac{1}{P_i} \right) \right)$$

Where:

- $PSSI(S_1, S_2)$ is the probabilistic distance between two strings S_1, S_2

- HD is the Hamming distance of a particular string alignment

- M is the number of misplaced character groups

- HD_{Δ} is the Hamming distance of the ith misplaced character group

- P_i is the probability of an error corresponding to the ith misplaced character group

For example:

PSSI(virusbtn, vrusbtn) = 1*1*log(1/0.5) = 0.301

PSSI(virusbtn, virusbtm) = 1*1*log(1/0.8) = 0.097

The greater the absolute value of $PSSI(S_1, S_2)$, the greater the distance between the two strings S_1 and S_2, and the lesser the chance that S_1 is mistyped as S_2 by a user. The whitelist described earlier can be used to calculate the *PSSI* value on the fly between the user-typed URL and the ones in the whitelist. The user can be warned if he types a URL with a *PSSI* value above a certain threshold.

CONCLUSIONS

We have discussed how probabilistic typo analysis techniques can be applied in the security domain – to prioritize the registering of unwanted domain names and to efficiently detect typographical errors entered into web browsers.

Along similar lines, email client plug-ins can be developed which can help prevent email typosquatting. This methodology can also be leveraged for parental control, wherein typos of popular children's websites can be generated on the fly, with the most likely typos added automatically to the restricted sites list of a browser.

Since anti-spam and anti-phishing techniques also involve the detection of typos of popular brands in the contained links, this technique could also play an important role in enhancing anti-spam and anti-phishing engines.

FEATURE 2

THE CHALLENGES OF COLLECTING AND MONITORING URLS THAT POINT TO MALWARE

André D. Corrêa
Malware Patrol, Brazil

The Malware Patrol project [1] is a fully automated and user-contributed system for collecting, analysing, blocking, alerting and monitoring URLs for the presence of malware. Since 2005 it has been cataloguing URLs used in phishing scams and distributing block lists for the most popular proxies and anti-spam systems. Every URL collected is visited daily to make sure the list is reliable and up to date.

FROM THE GROUND UP

In June 2005 a discussion took place on the GTS mailing list (a Brazilian security group hosted by Nic.br) regarding URLs pointing to malware. List members began submitting addresses, encouraging network administrators to block their users' access to them to prevent infections. It was no surprise that a couple of days later nobody knew which URLs were still active and what kind of malware they hosted. With this in mind, the Malware Patrol project was set up – initially operating under the name 'Malware Block List'.

Since the beginning, the goal of the project has been to provide a central point for the collection, analysis and monitoring of URLs pointing to dangerous file extensions, and also to distribute, free of charge to non-commercial users, up-to-date lists of infected addresses.

A rudimentary system was developed initially just to merge URLs and create a few block lists. It quickly became necessary to visit the addresses to make sure their status hadn't changed. Another important feature introduced during the first few weeks was the possibility of user contributions either via a web form or by forwarding suspect emails directly to a specially crafted mailbox.

EXTRACTING URLS

With the modernization and proliferation of phishing scams, fraudsters keep finding new and interesting ways to obfuscate the URLs included in their messages. URL obfuscation is indispensable as a means to trick users into downloading and installing malware. We find a vast variety of techniques being used to hide URLs, but there are also still a lot of 'newbie' phishers sending simple HTML email messages with URLs hidden by 'href' tags. Common obfuscation methods include using 'short URL' services, domain names that look legitimate except for the addition of a couple of letters or numbers, long URLs that begin with credible names, use of the '@' symbol (e.g. http://www.yourbank.com@example.com) and hexadecimal URL encoding. More advanced phishers are using JavaScript to create URLs during onClick(), multiple HTTP redirects, browser/crawler detection, Content-Disposition pages and fake anti-virus/anti-malware sites.

It is easy to envisage the technical and social engineering aspects of phishing scams evolving a lot more, becoming even more sophisticated and dangerous for end-users.

The constant evolution of obfuscation methods means that the project's URL extraction system must undergo constant development and improvement. One of the most recent threats that is forcing a profound change in the URL extraction system is the use of Content-Disposition [2, 3]. This feature is discussed in RFC 2183 (August 1997) and RFC 2616 (June 1999). Basically it is an extension of the MIME protocol that instructs the user agent on how to work with an attachment. Using Content-Disposition it is possible to point users to a non-dangerous URL (e.g. a PHP file) and from there send executable files or infected documents. It is an important security threat because proxies and content-filtering systems won't block users' access to PHP files, for instance, whereas they would deny access to EXE files.

This is a complicated issue for the Malware Patrol project due to the fact that URLs are filtered based on file extension. With a limited amount of processing power and bandwidth available there is a need to concentrate on dangerous URLs. Processing non-dangerous extensions, like PHP or ASP files, dramatically increases the CPU and bandwidth requirements.

YET ANOTHER BLOCK LIST

Common sense says we already have enough DNSBLs and block lists out there, so why bother running another block list? Well, the first 'Real-time Blackhole List' was created by Paul Vixie in 1997 [4]. Since then, DNSBLs and URL lists have become more sophisticated and focused on specific aspects, but none of them concentrate on URLs that point to malware. More importantly, most URL lists become outdated very quickly. There is a need for an accurate list of addresses that point to malware and the only way to maintain such a list is to visit every URL daily to verify its status.

Lots of factors can change the status of a URL. One of the most common situations involves malware being hosted on free web accounts. Such free accounts have a limited bandwidth allowance, and when that is exceeded access to the account is denied. Therefore, if a binary receives lots of hits it can quickly become unavailable, but will be reachable again in a few hours or the next day when the free hosting provider puts it back online.

Another important driver of status change is the renewal of malware from the hosting providers. Sometimes domain or hosting administrators act when alerted about malicious binaries on their servers and remove them. Among the most criticized issue of DNSBLs and URL lists is the slow response to removal requests. These requests are taken seriously by the Malware Patrol. Having a system that automatically validates all addresses in the database every day makes the removal process very fast and reliable.

Moreover, there are other good reasons to run a block list of malicious URLs including: current usage of Content-Disposition, distribution of binaries not yet classified by anti-virus solutions, alerting system administrators of malware hosted on their servers and cooperation with security groups.

NOT SO DANGEROUS EXTENSIONS

In addition to the common dangerous extensions everybody is blocking today, including but not limited to exe, pif, bat, scr, cmd, reg, com, etc., there are the ‘not so dangerous extensions’. Those cannot be blocked by default on proxies and content filters but pose serious threats to users. Examples include pdf [5], swf and png [6] files for which critical bugs were found recently. This is another strong motivation for using an up-to-date URL block list. These days, it is no longer possible to trust file extensions.

DETECTING NEW MALWARE

Approximately 20% of the samples collected by the project every day are pieces of malware that have not yet been classified by anti-virus vendors. This is an important threat for users that rely solely on anti-virus systems for protection. Currently, when a sample is not identified as malware by the anti-virus software we use, it goes through a scoring system. This system evaluates some important characteristics of the binary including: domain name and file name patterns, file extension, use of packers and cryptors, presence of strings, URLs or IRC commands, etc.

When a binary has a high score, the sample is sent to partner anti-virus vendors for analysis and classification. The project has already identified a lot of new malware. Most of the malware found on phishing sites are trojans targeting financial institutions.

Sandbox systems are not used for active malware analysis due to the complexity involved in automating these systems and the large amount of hardware resources required.

MALWARE VS. POC

Occasionally, some binaries and source code are crawled but their classification as malware is not obvious. It is important to differentiate malware targeting innocent users from software used for teaching or research purposes. In this category we can include: proofs of concept for vulnerabilities or bugs, hacktools, spamtools and even spyware. Although these can certainly be used for malicious purposes, they are not convenient for phishing scams. Proofs of concept are usually distributed in their source code format, needing compilation to run; hacktools and spamtools are attacking tools but have limited value when installed in a victim’s machine; and finally spyware may not be considered malware in the strict sense of the term.

Special attention must be paid to the rogue anti-virus, anti-spyware, anti-malware and scareware [7] that are proliferating on the Internet. Users are easily convinced to download and use them. Some of the current scams direct users to fake sites that appear to run an anti-virus scanner on the victim’s machine. The scanner reports numerous (non-existent) viruses or trojans on the system and claims that they can only be removed if the victim pays for the (rogue) software licence.

ALERTING AND REMOVING MALWARE FROM THE NET

When malware is found, it is important to notify system and network administrators and urge them to remove it from the Internet as soon as possible and to investigate the related criminal activity. This is done by the alert system that sends email messages to domain and ASN administrators and to the CSIRT responsible for the top level domain name of the domain hosting the malware.

It is disappointing that fewer than 15% of the alerts elicit a response. Although there are many Internet documents and RFCs [8] specifying the email addresses to be used by domain administrators, the majority of those addresses bounce, leaving no way of contacting the administrator.

Administrators should pay more attention and domain registrars could play an important role in educating and enforcing the existence of the abuse mailbox. This is necessary so that alerts and important information can be sent to administrators. Those mailboxes also need to be checked frequently – most of them bounce because they are full.

Many huge service providers take an annoying approach to abuse mailboxes. They simply respond with a default message pointing to a web form that must be filled in to file a complaint. Although it is understandable that enormous volumes of messages (mostly spam) are sent to those addresses, this approach doesn't work when security groups are trying to cooperate and it is not permitted by any RFC.

After a few weeks of running the malware alert system an unexpected side effect was noticed: the alert mailbox started receiving spam and phishing scams.

COOPERATING WITH THE SECURITY COMMUNITY

The cooperation and exchange of information with security groups and professionals is essential for any project that captures and redistributes data. The Malware Patrol is open to establishing working relationships with any serious security group that can send and/or wants to receive information regarding malware and phishing scams. Some of the groups that already cooperate with the project include: CAIS - RNP [9], Team Cymru [10], Web of Trust [11], SURBL [12], the now ceased CastleCops project, as well as more than 10,000 personal contributors.

Although the project has many spam traps established, contributions from users and security professionals are also very important to capture the most recent phishing scams. There are two contribution channels available: a web form that can be filled in with suspect URLs and an email address to which suspect messages can be sent, which extracts URLs and puts them in a queue for later analysis. The exchange of information with security groups is mostly done via email.

Other ways of acquiring URLs that are used or are under development include: monitoring mailing lists and newsgroups related to information security, botnets and hacking, web crawling and IRC monitoring bots.

KEEP CRAWLING

New URLs go to a queue that is visited by an automated crawler every hour. The crawler visits the addresses and grabs binaries, if available. If no binary is found, it verifies the HTTP status code and HTML output generated by the web server. The crawler can follow many levels of redirection via HTTP and HTML and can also handle most of the Content-Disposition scenarios.

Every URL in the database has a status that varies from infected to not available, not found, invalid Content-Type and others. Those addresses are visited every day to check that their status hasn't changed. The only way to keep the block lists accurate is to frequently visit all addresses. There is also a need to impersonate different browsers because some fraudsters use browser detection to prevent crawlers from grabbing their binaries.

BLOCK LISTS FOR EVERYONE

For end-users, the most important information produced by the project is the block lists. Today we have 29 different formats available for non-commercial use. They include popular open-source software like: *Squid*, *SpamAssassin*, *Postfix*, *Firekeeper*, *DansGuardian* and even *ClamAV*. There are two types of list: the 'regular' list, which includes protocol, host name, domain name and directories; and the 'aggressive' list that just includes protocol, host name and domain name, therefore blocking access to the entire infected domains.

File names and extensions of malware are never disclosed to end-users. The project also refuses to exchange malware samples without a really good reason to do so.

THERE ARE 60 MINUTES IN EVERY HOUR

Most list downloads are made by automated scheduled jobs and it is easy to figure out that administrators like the minute zero of every hour to run those jobs. This causes a tremendous overhead on web servers during the three or four minutes before and after the turn of the hour. It is strongly advisable to set schedules to run at other times when the system is running lower on CPU usage. This also helps to prevent download timeouts that can lead to broken or incomplete block lists.

SERVERS AND SOFTWARE

Excluding the anti-virus scanners used to identify and categorize malware, all other software employed by the project is open source. Servers are hosted in three distinct data centres and run *Linux Slackware* [13] or *FreeBSD* [14]. *MySQL* is used for the database. The crawlers run multi-threaded on Perl and C, *Apache* and *LIGHTTPD* are used for web servers and *Exim* for SMTP, running the URL extraction system.

WHAT'S NEXT?

Malware Patrol is a not-for-profit project that runs on volunteer efforts, user contributions and donations. With limited hardware, software and financial resources, there is a need to concentrate efforts on service availability and data integrity. Meanwhile, the improvements queue is always growing. Important developments scheduled for the coming months include: installing new hardware to support the growing number of users, integration with browser plug-ins, creation of a DNSRBL, improvements to crawlers and URL extractors, and creation of thumbnails of the emails and sites used in phishing scams.

New list users, contributors and security groups are always welcome.

Also, users and companies that find the service useful are encouraged to make a donation to help keep the project going.

CONCLUSION

For the last four years the Malware Patrol project has been collecting, monitoring and distributing lists of URLs that point to malware. The project provides an additional security tool for system and network administrators in the daily task of keeping users safe. End-users and the security community are always responsive to reliable tools and information sources.

The phishing scam landscape is constantly evolving and although anti-virus solutions and user education are important ways to help prevent infections and the losses caused by malware, it is also necessary to closely monitor the evolution of these threats.

REFERENCES

[1] Malware Patrol. http://www.malwarepatrol.net/.

[2] The Content-Disposition Header Field. http://www.ietf.org/rfc/rfc2183.txt.

[3] Hypertext Transfer Protocol - HTTP/1.1. http://www.ietf.org/rfc/rfc2616.txt.

[4] DNSBL - Wikipedia. http://en.wikipedia.org/wiki/DNSBL.

[5] Adobe - Security bulletins and advisories. http://www.adobe.com/support/security/.

[6] Multiple vulnerabilities in libpng. http://www.us-cert.gov/cas/techalerts/TA04-217A.html.

[7] Wikipedia - Scareware. http://en.wikipedia.org/wiki/Scareware.

[8] Mailbox names for common services, roles and functions. http://www.ietf.org/rfc/rfc2142.txt.

[9] CAIS - RNP. http://www.rnp.br/cais/.

[10] Team Cymru. http://www.team-cymru.org/.

[11] Web of Trust. http://www.mywot.com/.

[12] SURBL. http://www.surbl.org/.

[13] Linux Slackware. http://www.slackware.com/.

[14] FreeBSD. http://www.freebsd.org/.

PRODUCT REVIEW

NORMAN NETWORK PROTECTION APPLIANCE

John Hawes

This month's product review marks something of a departure for *VB*, as we take a break from our usual diet of desktop anti-virus products and security suites to take a look at one of the growing field of hardware-based security solutions. The security appliance market seems to have become a boom area of late, with just about every security firm worth its salt introducing an appliance solution to provide its services in a single package. Dedicated firewalls, spam filters and web filters, as well as integrated blends offering a selection of these features, all jostle for position in the marketplace, all looking for the unique selling point which will set them apart from the crowd. In this month's review we will be looking at a dedicated anti-malware solution: *Norman's Network Protection Appliance*.

PRODUCT AND VENDOR

Norman has been around since pretty much the dawn of time as far as computer security and anti-malware goes. Founded in 1984 (as the company's website points out, some two years before the first PC virus and four years before the first virus was discovered in the company's native Norway), the company quickly went global and over the next decade or so developed a variety of security solutions and services, before merging with the brains behind the classic *ThunderByte* anti-virus product. *Norman*'s *Virus Control* product line has been a stalwart in VB100 testing since its inception over ten years ago – one of an elite few to achieve a pass in the first ever certification (see *VB*, January 1998, p.10) and maintaining a splendid reliability ever since.

Alongside its venerable flagship product line, the company has continued to innovate in a number of areas, with pioneering work at corporate server and gateway level in the early days, followed up by early entry into the spheres of personal firewalls and anti-spam. The *Sandbox* solution – which evolved from internal emulation in the *Norman* scanner into a standalone, automated malware analysis system – was a revolution when it first appeared and remains cutting edge (as anyone who has joined a crowd of entranced onlookers at a demonstration of its abilities can attest). The inclusion of the advanced *Sandbox* technology in the company's anti-malware scanner, and the resulting high levels of signature-less detection of new malware, have made the engine a favourite for inclusion in OEM and multi-engine products, many of which are now also regulars in *VB* testing.

The company's online presence at www.norman.com has undergone a fairly drastic overhaul of late, and provides a wealth of information in a slick and accessible format. The *Network Protection* solution is fairly new to market and is currently being heavily promoted, with a prominent advertisement in the form of an animation showing a machine sliding rapidly into place – apparently a metaphor for the system's easy implementation. A selection of related information is provided on the website, including registration for a trial version, detailed product overview sheets and a full manual. A link provides details of a recent accolade for the product: a 'Best in Antimalware Solution' prize from *Network Products Guide*.

The appliance is available either pre-installed, with a choice of fairly standard set-ups of either *Dell* or *HP* hardware, or as a software version, provided complete with operating system as a CD or iso image download. For those opting to use their own hardware, the minimal specifications are fairly exacting, thanks to the rather specific demands of the networking and RAID design and the need for reliability. Thus the software version is unlikely to suit anyone trying to save pennies by recycling an old system, but the self-install option is provided for those organizations with fixed policies on hardware sourcing; in addition to the *HP* and *Dell* specs also available directly from *Norman*, an acceptable design of *IBM* hardware is approved.

The version provided for this review was the full hardware set-up, installed on a *Dell* box, so we didn't get to try the complete installation process, but as a pre-configured system it seemed likely to be fairly straightforward; the guidance in the manual mainly covers correct implementation of network cards, with the bulk of the set-up on a fixed, option-free path.

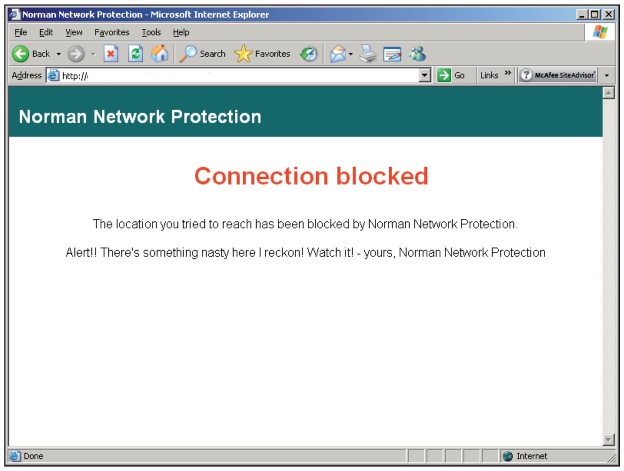
The shipped machine came with a clear and straightforward quick-install guide, which looked ample to steer us through, but we also took along a copy of the manual in case of need. The manual is provided both as a complete guide and as a tailored version for purchasers of the full hardware appliance, leaving out all the unnecessary information on initial installation. With plenty of support on hand, we took the box into the test lab and prepared to fire it up.

INSTALLATION AND SET-UP

Following the instructions in the admin guide, a network layout was planned but nothing plugged in until the initial stages of IP address configuration were completed. During the initial stages of our anti-spam testing regime we spent many hours setting up a selection of appliance products, and experienced a wide variety of initial set-up processes required to get a machine activated and integrated into a network. It was interesting to see just how many different ways there are of getting a piece of hardware configured out of the box – some require full direct access with a keyboard and monitor, others demand access via a pre-configured IP address, while a few are even accessed via a good old-fashioned null-modem cable. At least one appliance on the market is capable of picking up an address from DHCP and displaying it on an LED front panel, allowing the administrator to access it with no special tweaks to his network. In *Norman*'s case, the initial stage required console access with a keyboard and monitor, which should present little difficulty to the average corporate admin with a KVM switch handy.

The boot-up showed the system to be based on *Debian Linux*, with a slick and attractive splashscreen identifying it as a *Norman* product. With the initial stages of booting complete, a simple text-based process led us through the low-level configuration: the keymap used, a password for the root user, hostname and IP address for the management interface, speed configuration of the additional interfaces, and time zone. With this complete and details confirmed, the machine was rebooted before being ready for remote control via the management link. The manual urges readers not to boot up the machine before connecting it to the network, but being dedicated troublemakers we ignored this advice and tried various configurations of switching on and off and plugging network cables in and out, without upsetting the system at all. Of course, such behaviour is not recommended and we would urge readers always to pay attention to instructions from their solution providers.

Once up and running again, and connected to the chosen management network, a web interface is accessed via a browser, which is fairly standard for appliances. This one seemed smooth and stable, with none of the awkward load time or wobbly, error-prone controls we have observed in some of the appliances run as part of our anti-spam

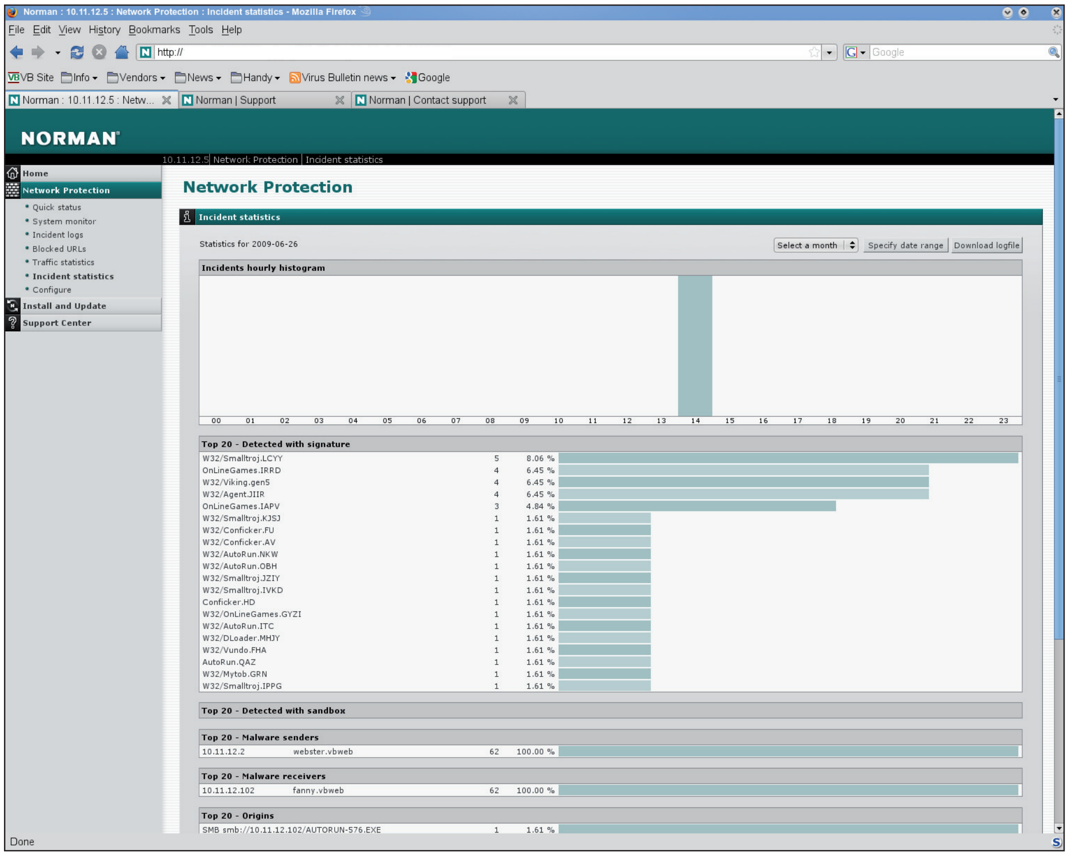

Norman Network Protection - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://
Go Links McAfee SiteAdvisor
Done
Internet

Norman Network Protection

Connection blocked

The location you tried to reach has been blocked by Norman Network Protection.

Alert!! There's something nasty here I reckon! Watch it! - yours, Norman Network Protection



Network Protection

Incident statistics

Statistics for 2009-06-26

Top 20 - Detected with signature

W32/Smalltroj.LCYY	5	8.06 %
OnLineGames.IRRD	4	6.45 %
W32/Viking.gen5	4	6.45 %
W32/Agent.JIIR	4	6.45 %
OnLineGames.IAPV	3	4.84 %
W32/Smalltroj.KJSJ	1	1.61 %
W32/Conficker.FU	1	1.61 %
W32/Conficker.AV	1	1.61 %
W32/AutoRun.NKW	1	1.61 %
W32/AutoRun.OBH	1	1.61 %
W32/Smalltroj.JZIY	1	1.61 %
W32/Smalltroj.IVKD	1	1.61 %
Conficker.HD	1	1.61 %
W32/OnLineGames.GYZI	1	1.61 %
W32/AutoRun.ITC	1	1.61 %
W32/DLoader.MHJY	1	1.61 %
W32/Vundo.FHA	1	1.61 %
AutoRun.QAZ	1	1.61 %
W32/Mytob.GRN	1	1.61 %
W32/Smalltroj.IPPG	1	1.61 %

Top 20 - Detected with sandbox

Top 20 - Malware senders

10.11.12.2	webster.vbweb	62	100.00 %

Top 20 - Malware receivers

10.11.12.102	fanny.vbweb	62	100.00 %

Top 20 - Origins

SMB smb://10.11.12.102/AUTORUN-576.EXE	1	1.61 %

testing. On initial access, the appliance runs through a second stage of configuration, this time focusing on the protection side of things, starting with the selection of a password and configuration of access controls for the configuration interface itself, restricting access to specific IP addresses or subnets.

A licence key is required for activation, and then a selection of settings screens are run through, including a separate setting for each of the protocols covered by the system. The protocols covered include HTTP, FTP, TFTP, SMTP, POP3, SMB/CIFS, RPC and IRC, and each can be scanned at a different level, with traditional signature scanning, the sandbox, and looking inside archives all available. Other options include how long to keep known bad URLs blocked, and a configurable message to display to users trying to access a blocked site. This can take the form of either a customized message on a standard web page or a bespoke web page at the location of your choice. Some control of logging and email alert messaging is also provided, and updating can be set to a range of periods or left entirely manual. All in all, barring the time spent getting hold of a licence key to activate – which would normally be provided along with the shipped product – and despite some time spent messing around with settings and trying unsuccessfully to confuse the system, the whole set-up from initial boot to fully operational status took less than half an hour, much of which was spent waiting for reboots and rummaging for network cables of the right length.

MALWARE PROTECTION AND CONTROL

With the machine set up more or less to our liking, it was time to see how it went about protecting networks. The design of the product is brilliantly simple; apparently inspired by a demanding commission from a food manufacturer (requiring protection from malware in a sealed and certified environment where changes to either software or network configuration were highly undesirable), the *Norman* appliance sits invisibly between two network nodes, its two interfaces simply passing all data through and keeping an eye on the stream as it goes by, blocking the transfer of anything identified as a danger. So we simply slid the machine in between the hubs of two subnets, moved the cable connecting them to one interface on the appliance, inserted another in the second interface to complete the link, and sat back to watch.

After an invisible judder as the network adjusted itself to the new layout, connection between the two subnets seemed entirely unaffected and data transfer between them continued virtually uninterrupted. Checking the management GUI showed that traffic was being watched and throughput levels recorded, and attempts to pass malware samples from the outer zone to the protected subnet were immediately blocked. It all seemed very easy and painless.

We tried a variety of transfers via various protocols, and all seemed to function along much the same lines, with network latency barely noticeable as traffic flowed smoothly through the system. Even with large files and the most thorough settings, little slowdown was observed, and malicious items tucked into archives were noticed as the final few packets made their way across the network. Compared to more proxy-oriented appliances, which may require full download to the appliance before scanning, then transfer to the target system once files have been approved, this invisible bridging set-up provides much smoother and quicker connectivity. The system automatically blocks access to web domains found to contain malware, showing a warning message instead, while other protocols simply present a 'not found' or 'access denied', again keeping the specific path blocked for a configurable period. If desired, the URL blocked message shown for the HTTP protocol could even be configured to display a standard 404 message, thus erasing all evidence from the user's point of view that the appliance is monitoring traffic to their network.

Going back into the configuration system to tweak settings provided the same set of controls run through in the initial set-up, with the main area being the level of scanning imposed on each supported protocol, with the option to ignore all traffic over a given protocol if so desired. The most significant option here is whether or not to use the *Sandbox* facility. We found this gave a significantly better level of detection, particularly for the newest samples, without excessive increase to the connection overhead. There is also a simple way to exclude certain systems or network segments from filtering, and a set of options that allow scanning to be disabled, passing all traffic through unhindered, or to completely block all traffic – the so-called 'panic button'.

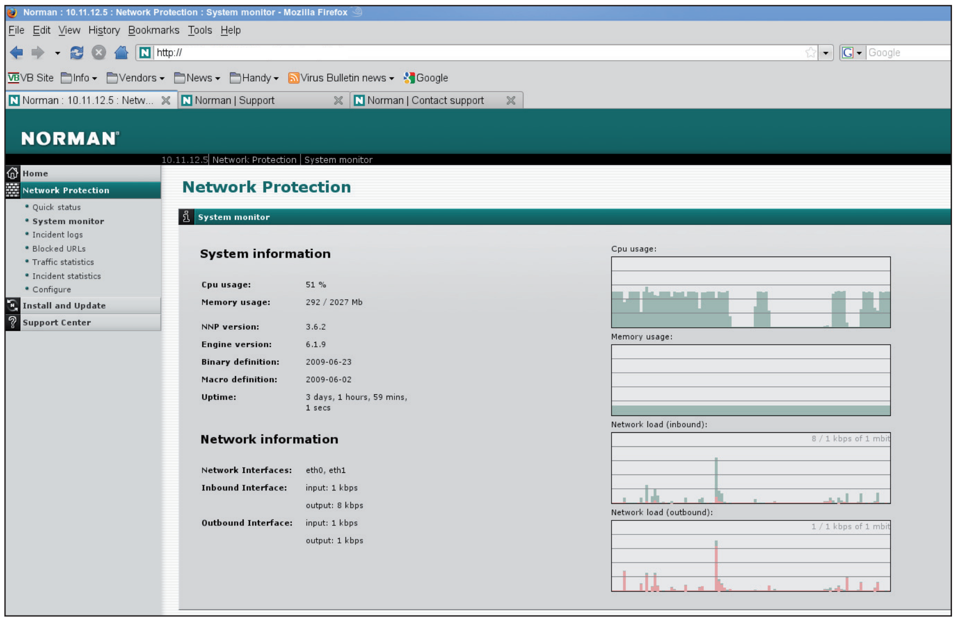
The remainder of the controls provided in the interface covered the logging and reporting of traffic and incident data. One area that seemed to be missing was the option to adjust the low-level settings of the appliance operating system itself, but little needs doing here in most situations. Rather oddly, when we went back in via the console and hacked some changes to the IP address to connect to a different network for management, we found that the web GUI did not register the change but continued reporting the old address (perhaps because we hadn't restarted the system), but no ill effects were observed and everything continued to operate smoothly.

REPORTING FEATURES

The main function of the GUI, once the basic settings have been adjusted as required, is to provide information on traffic and incidents, and it provides a useful selection of tables and graphs which present all the necessary information clearly and logically.

The 'Quick status' page provides an overview of the system set-up and settings, details of the product and update versions, a summary of the network interfaces and how they are running, the number of files processed and those found to be malicious, and the settings of the scanner on a per-protocol basis. The 'System monitor' gives more detailed information on CPU and memory usage, uptime, product and update versions, and network load, accompanied by some nice graphs which chart changes in load over time. The other sections show statistics on traffic passing through the system, and on threats spotted and blocked, with detailed logs of all detected threats and malicious URLs, and full data on systems hosting and targeted by malware. All of this information can be configured to cover specific periods and levels of verbosity, and exported to plain logs.

One of the most interesting features here is the *Sandbox* log area. Where a malicious item has been run through the *Sandbox* system, detailed information is logged on the behaviours spotted when executed in the emulator. This data, including information on how a file has been packed or encrypted, what changes it makes to the filesystem, what network activity it attempts and more besides, is also made available to the administrator. These reports always make for fascinating reading, and are of great value in identifying



Network Protection

System monitor

System information

Cpu usage:	51 %
Memory usage:	292 / 2027 Mb
NNP version:	3.6.2
Engine version:	6.1.9
Binary definition:	2009-06-23
Macro definition:	2009-06-02
Uptime:	3 days, 1 hours, 59 mins, 1 secs

Network information

Network Interfaces:	eth0, eth1
Inbound Interface:	input: 1 kbps
	output: 8 kbps
Outbound Interface:	input: 1 kbps
	output: 1 kbps

malicious items and tracking down any activities they may have perpetrated before being caught. With automated log retrieval and parsing, the data can be used to keep other parts of the network secure by updating firewall rules and other security systems.

Email and SNMP options allow alerts to be sent to administrators without keeping an eye on the interface, again with fairly in-depth configuration of what level of data is recorded, where it is sent and how. With the logs saved on the local system, administrators can of course also automate transfer of logs as required, using the underlying fully functional *Linux* system. This would allow a fairly detailed record of all activity passing between the appliance's bridged interfaces, including what kind of malware was coming in, from where, and where it was headed when intercepted – all useful data for the alert security admin.

CONCLUSIONS

Having looked at a number of similar appliances in recent months, there were a few things that we expected to find in *Norman*'s solution but which were lacking. Many products targeting the same sort of market offer online reputation databases to block known malicious URLs, detailed content inspection and control of user activity, for corporations to keep a tight rein on their employees' web-browsing habits, configurable monitoring of specific applications and versions, and much else besides. *Norman*'s selling point is all about simplicity though. Its raison d'être is to block malware passing through the core protocols, and it does that with remarkably little effort on the administrator's part. The plug-and-play ease of implementation also makes it very flexible, happy to be installed in any part of the network rather than being limited to the gateway.

Being practically undetectable to the network it is protecting, up until the moment it blocks the transfer of a malicious item, makes it not only extremely easy to integrate into just about any network layout, but it also keeps the overheads to a minimum, barely impacting traffic flow until it is needed. The integration of the *Sandbox* detection alongside the traditional signature scanning adds an extra layer of defence against new and unknown threats. Providing such simple and unproblematic malware protection, along with an excellent, again very straightforward control system, makes this an extremely user-friendly weapon in the fight against malware problems in business networks, as well as a powerful one. We look forward to investigating a wider range of appliances to see how they match up to this impressive effort from *Norman*.

COMPARATIVE REVIEW

ANTI-SPAM COMPARATIVE REVIEW JULY 2009

Martijn Grooten

VB's first anti-spam comparative review and certification showed some interesting results (see *VB*, May 2009, p.S5), and the winners of VBSpam awards deserve full credit for doing so well. This month, the all-important question is: can they repeat their outstanding performance?

Prior to the first anti-spam comparative review we ran a trial, during which the licence for one of the participating products expired. This product was configured to continue to work, yet the anti-spam engine was no longer being updated – indeed, when we looked at its performance, the spam catch rate gradually decreased over time. Apart from acting as a reminder of how important it is to renew licences, this served as a demonstration that spam changes over time and that anti-spam vendors need to ensure their engines are up to date and work against the latest spam threats. This is why we run a new anti-spam test every two months.

Nine products participated in this month's test, seven of which were commercial products, while the other two were free and open source. Of the seven commercial products, two were hosted solutions, two were hardware appliances, one ran on a *Windows Server* machine and the final two ran on *Linux*. Together they provide a good representation of the range of different options available when it comes to spam filtering within an organization.

THE TEST SET-UP

The set-up of this test was more or less the same as for the last test. The methodology is recorded at http://www.virusbtn.com/vbspam/methodology.

Some changes to the MTAs enabled us to run a test with a much larger email corpus and more products running in parallel; however, these changes do not affect the test set-up itself. As in the previous test, the Message ID was used to uniquely identify emails, and if such a header was not already present, the gateway MTA added one. Email that did not reach the back-end server within an hour was assumed to have been classified as spam. During this hour, up to five redelivery attempts were made for emails that had caused an error during the SMTP transaction.

Unlike in the previous test, the two products that ran on the gateway MTA, *SpamAssassin* and *ClamAV*, were sent emails in real time.

The products that needed to be installed on a server were installed on a *Dell PowerEdge R200*, with a 3.0GHz dual core processor and 4GB of RAM.

THE EMAIL CORPUS

As before, all of the emails sent to valid addresses on the virusbtn.com domain were sent through all of the products, in real time. This corpus consisted of 2,393 ham messages and 26,755 spam messages. The 'golden standard' for each email was decided upon by the recipient, with the exception of emails for which all products agreed: in these cases we assumed the products were correct. Emails that were reported as false positives were checked a second time, to make sure none had been misclassified by the recipient as ham.

To increase both the volume and the variety of spam seen by the products, we have been working closely with *Project Honey Pot*. The *Project Honey Pot* team manages a large distributed network of spam traps and thus receives not only a large amount of spam, but also spam that reflects the global variation in bogus email. For this month's test they sent us part of their feed; the emails were assigned to a random valid address on the virusbtn.com domain and then relayed to the products in real time.

So that the products would not have any information about which emails were part of this feed (all of which, of course, were assumed to be spam) the Received-headers were rewritten, so that it appeared as if the email had been received by our MTA. Moreover, many spam emails are 'personalized' and thus contained the local-part and/or the domain of the original spam trap; these were replaced by the local-part of the newly assigned recipient and virusbtn.com respectively.

The *Project Honey Pot* feed provided us with 716,256 additional spam emails, which meant an overall corpus of 745,404 emails, 2,393 of which were ham. As the test ran for a period of almost three weeks (starting at 16:45h on 5 June and finishing at 08:00h on 26 June) this meant that the products saw about 25 emails per minute or almost one email every two seconds.

It was interesting to see that all products performed better against the *Project Honey Pot* spam than against the *VB* spam; in most cases the difference in performance was rather significant. We can only guess the reason for this, but it could well be caused by the nature of the spam *VB* receives: like most companies, we receive a large amount of commercial email, unsolicited and sent in bulk, yet somehow targeted. Emails like these don't generally end up in spam traps and do not look like 'typical' spam, yet they are illegal and filters should be expected to block them.

FALSE POSITIVES

Anyone comparing the false positives reported in our test with those reported by other anti-spam tests or by the vendors themselves, will notice that our numbers are significantly higher. There are several reasons for this.

Firstly, the emails received by *VB* employees – many of which discuss malware and spam – are particularly hard to filter. For example, we may see a legitimate message in which a spam domain is being discussed, and this message is classified as spam by one or more products on the premise that it contains this very domain. While the mistake is understandable, these messages are ham and an ideal spam filter would not make this mistake.

Secondly, most products give the end-user and/or the system administrator the option to whitelist certain senders. While we encourage the use of such techniques in practice, we have not applied them in our tests: it would be hard, if not impossible, to perform whitelisting in the same way as an average end-user, plus it would put any products entering the test for the first time at a disadvantage.

This is one of the main reasons why our certification scheme emphasizes the relative performance of products compared to those of their competitors, rather than focusing on actual numbers.

In the results reported below, 'SC rate (total)' represents the overall spam catch rate over the entire corpus of 745,404 emails. 'SC rate (Project Honey Pot corpus)' represents the spam catch rate achieved within the Project Honey Pot corpus alone, and 'SC rate (VB spam corpus)' represents the spam catch rate achieved within the VB corpus alone. 'FP rate' represents the number of false positives as a proportion of the total number of ham messages, while the 'FP rate of total VB mail corpus' is the number of false positives as a proportion of the total number of messages contained in the VB mail corpus.

BitDefender Security for Mail Servers 3.0.2

SC rate (total): 98.23%

SC rate (Project Honey Pot corpus): 98.56%

SC rate (VB spam corpus): 89.36%

FP rate: 2.55%

FP rate of total VB mail corpus: 0.209%

Romanian company *BitDefender* won a gold VBSpam award for its *Linux* server product in the May 2009 test and its developers were keen to see if the product could repeat its performance. A fresh installation of the product, again as an extension to *Postfix*, but this time installed on a server running *SUSE11*, was set up easily.

The spam catch rate of more than 98% is a huge improvement compared to the previous test and even the product's performance on the *VB* spam corpus increased by more than five per cent. Unfortunately, the false positive rate also increased to a level that just pushed the product outside the limits for a gold award; instead it wins a VBSpam Silver award this month.

ClamAV using Sanesecurity signatures

SC rate (total): 73.97%

SC rate (Project Honey Pot corpus): 75.04%

SC rate (VB spam corpus): 45.51%

FP rate: 0.38%

FP rate of total VB mail corpus: 0.031%

The signatures provided by *Sanesecurity* that can be plugged into the open source *ClamAV* anti-virus product were never meant to match the performance of dedicated anti-spam solutions. Rather, they are intended to work together with another solution to provide a layered spam filter. Still, the 23% spam catch rate measured in May was disappointing for the developer despite a zero false positive rate.

The developer will be happy to hear that this month the product's performance almost tripled to 74%, and even on the *VB* spam corpus it increased by 18% – and although these rates are insufficient to earn the product a VBSpam award, they indicate that the product is well up to its task as the first step in a multi-layered spam filter. There were a handful of false positives this time, all of which were caused by legitimate emails mentioning a malicious domain or email address – and it would be fair to say that it would be rare for the majority of end-users to receive such emails.

FortiMail

SC rate (total): 99.11%

SC rate (Project Honey Pot corpus): 99.28%

SC rate (VB spam corpus): 94.38%

FP rate: 2.63%

FP rate of total VB mail corpus: 0.216%

Fortinet, based in Vancouver B.C., is a regular participant in the VB100 anti-malware testing, so it came as little surprise that the company was eager to submit its *FortiMail* appliance for the anti-spam test. As with most hardware appliances, no software needed to be installed and after a short set-up process, the product was up and running.

Further configuration, to fine tune the appliance, can be carried out via a web interface, while another web interface can be used by end-users to view their quarantined email or modify per-user settings. The web interface isn't used purely to click buttons however: one of the most important tasks of a system administrator running an anti-spam solution is to find out why certain emails were blocked and to prevent this from happening again. Those using *Fortinet* will have an easy task doing so, thanks to an extensive logging system: for every email received the logging system records which anti-spam tests were passed or failed.

With a stunning spam catch rate of 99.1%, *FortiMail* outperformed all of its competitors in this respect, and even on the *VB* corpus well over 94% of the spam was identified correctly. On the downside, the product's false positive rate was slightly higher than average and thus *FortiMail* debuts by winning a VBSpam Silver award.

Kaspersky Anti-Spam 3.0

SC rate (total): 97.54%

SC rate (Project Honey Pot corpus): 98.17%

SC rate (VB spam corpus): 80.81%

FP rate: 0.04%

FP rate of total VB mail corpus: 0.003%

Kaspersky is another VB100 regular that has joined the anti-spam test this month. The Russian anti-malware giant has been active in the anti-spam field for a long time and the product we tested is the third generation of *Kaspersky Anti-Spam* (*KAS*).

A *Linux* product, we ran it on a *SUSE11* server as an extension to the *Postfix* MTA. Installation was smooth and painless and the product was running just a few minutes after the download had finished. After that, its performance gave so few reasons to worry that not until I started writing this review did I have a reason to search for log files, upon which I happily discovered that the decision made for each email is indeed stored.

Administrators running *KAS* will have little reason to look in these log files for false positives though: out of almost 2,400 ham messages sent to the product, only one was marked as spam. This unbelievably low false positive rate combined with a spam catch rate of over 97%, well above average, means that *Kaspersky Anti-Spam* is the deserving recipient of a VBSpam Platinum award.

MessageStream

SC rate (total): 98.82%

SC rate (Project Honey Pot corpus): 99.21%

SC rate (VB spam corpus): 88.48%

FP rate: 1.59%

FP rate of total VB mail corpus: 0.130%

You would be forgiven for thinking that vendors submit their products for the anti-spam test purely for marketing reasons, but many developers are also keen to hear our feedback so that they can see how and in which areas their products can be improved. UK-based *Giacom*, which develops the hosted solution *MessageStream*, achieved a VBSpam Gold award in May, yet felt that its false positive rate could be reduced, resulting in its developers making some changes to its filtering for all of its customers.

These changes were rolled out halfway through this month's test, but the number of false positives was already reduced to about 1.5% – well below average. This, together with a spam catch rate of almost 99%, means that *MessageStream* has achieved its second VBSpam Gold award in a row.

ModusGate

SC rate (total): 95.41%

SC rate (Project Honey Pot corpus): 95.63%

SC rate (VB spam corpus): 89.48%

FP rate: 6.64%

FP rate of total VB mail corpus: 0.545%

As someone with more than a decade of experience with various flavours of Unix and *Linux* I sometimes think that *Windows* software works against my intuition. This prejudice, however, was quickly proven wrong when I installed *ModusGate*, an anti-spam solution produced by Canadian company *Vircom*. The product is available as a hardware appliance, but tested here as a software solution installed on a *Windows Server 2003* machine. Set-up and installation were straightforward and the graphical interface was clear and easy to work with.

Unfortunately, despite a decent spam catch rate, the product's false positive performance was disappointing, with a score of more than 6%. The majority of the misclassifications were emails that had been sent as mass-mailings. Regrettably, the product's high false positive rate was enough to deny it a VBSpam award in this month's test.

M+Guardian (Messaging Architects)

SC rate (total): 98.92%

SC rate (Project Honey Pot corpus): 99.12%

SC rate (VB spam corpus): 93.63%

FP rate: 0.79%

FP rate of total VB mail corpus: 0.065%

Messaging Architects was one of the first companies to submit its solution, *M+Guardian,* for the anti-spam test and the confidence its developers have shown in their product proved to be justified: the product was the sole winner of a VBSpam Platinum award in the first test. It was with interest that we assessed the results this month to see whether the product would be able to continue its excellent performance in the second test, with more spam, more products and stricter benchmarks.

Happily for *Messaging Architects* it did continue its excellent performance. A spam catch rate of almost 99% and a less than 1% false positive rate earn *M+Guardian* its second VBSpam Platinum award.

SpamAssassin 3.2.5

SC rate (total): 64.26%

SC rate (Project Honey Pot corpus): 64.72%

					Total spam			Project Honey Pot corpus			VB corpus		
	True negative	FP	FP rate	FP / total VB corpus	False negative	True positive	SC rate	False negative	True positive	SC rate	False negative	True positive	SC rate
BitDefender	2332	61	2.55%	0.209%	13133	729878	98.23%	10285	705971	98.56%	2848	23907	89.36%
ClamAV	2384	9	0.38%	0.031%	193391	549620	73.97%	178813	537443	75.04%	14578	12177	45.51%
FortiMail	2330	63	2.63%	0.216%	6640	736371	99.11%	5136	711120	99.28%	1504	25251	94.38%
Kaspersky	2392	1	0.04%	0.003%	18268	724743	97.54%	13134	703122	98.17%	5134	21621	80.81%
MessageStream	2355	38	1.59%	0.130%	8752	734259	98.82%	5670	710586	99.21%	3082	23673	88.48%
ModusGate	2234	159	6.64%	0.545%	34128	708883	95.41%	31313	684943	95.63%	2815	23940	89.48%
M+Guardian	2374	19	0.79%	0.065%	8006	735005	98.92%	6302	709954	99.12%	1704	25051	93.63%
SpamAssassin	2324	69	2.88%	0.237%	265516	477495	64.26%	252664	463592	64.72%	12852	13903	51.96%
Webroot	2335	58	2.42%	0.199%	25659	717352	96.55%	24310	691946	96.61%	1349	25406	94.96%

SC rate (VB spam corpus): 51.96%
FP rate: 2.88%
FP rate of total VB mail corpus: 0.237%

The ancient, but still actively developed, open source *SpamAssassin* product took part in the first anti-spam test, but failed to match the performance of most of its commercial competitors. It was suggested that this was because an old version of the product had been installed. With the latest version, 3.2.5, running on a fresh *SUSE11 Linux* server, we were keen to see if this would have a positive effect on its performance.

Unfortunately, the spam catch rate was still lower than 65%, while the false positive rate actually rose to a higher level than previously. While the *sa-update* command, which is run every hour, suggests that nothing is the matter, it seems likely that the poor results are caused by an incorrect configuration, one that can hopefully be fixed before the next test.

Webroot E-Mail Security SaaS

SC rate (total): 96.55%
SC rate (Project Honey Pot corpus): 96.61%
SC rate (VB spam corpus): 94.96%
FP rate: 2.42%
FP rate of total VB mail corpus: 0.199%

Webroot's hosted solution failed to win an award in the May test because its number of false positives was a lot higher than the benchmark – a problem caused by an incorrectly configured SPF test. The problem was easily fixed, however, and indeed, the false positive rate of 2.42% measured in this month's test is only slightly above the average.

vb July 2009 silver SPAM virusbtn.com

The product also achieved a very decent 96.5% spam catch rate, and deserves extra credit for having the highest spam catch rate on the *VB* spam corpus. A VBSpam Silver award is thus well deserved and only a slight improvement of the false positive rate would enable it to do even better next time.

AWARDS

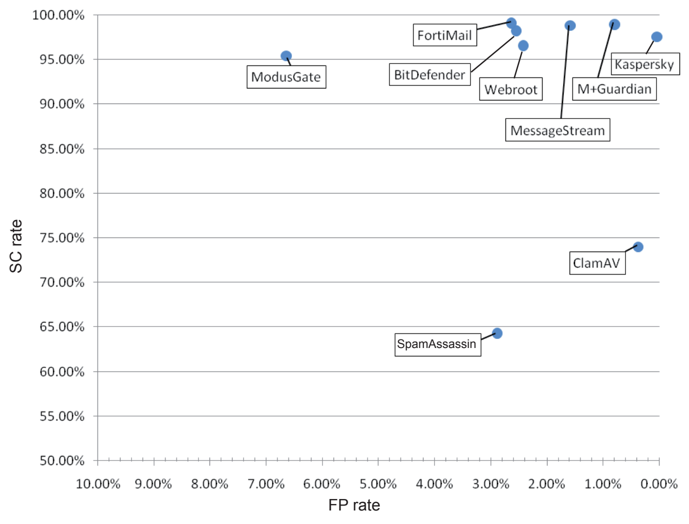
As in the previous test, the level of the awards earned by products are defined as follows:

- VBSpam Platinum for products with a total spam catch rate twice as high and a false positive rate twice as low as the average in the test.
- VBSpam Gold for products with a total spam catch rate at least as high and a false positive rate at least as low as the average in the test.
- VBSpam Silver for products whose total spam catch rate and false positive rates are no more than 50% worse than the average in the test.

To avoid the averages being skewed by one or more malperforming products, any product with a false positive rate of more than 10% and/or a spam catch rate of less than 70% is removed from the computation of the averages. In this case, the *SpamAssassin* scores were removed, because its spam catch rate was well below 70%. This month's benchmarks were then as follows:

- Platinum: SC 97.41%; FP 1.07%
- Gold: SC 94.82%; FP 2.13%
- Silver: SC 92.23%; FP 3.20%

The table opposite shows the scores for all of the products on test. The highlighted columns show the scores used for the benchmark calculations.



CONCLUSIONS

It was a relief to see the bugs that had caused some stress during the first test had been solved. It is also good to see several of the products showing credible results and thus to get a better picture of which products really are the high performers in this field. Still, the question remains as to whether these products can continue their performance in the next test, and it will be interesting to see the effect of the improvements and tweaks that will undoubtedly be made to other products.

The next anti-spam comparative review will take place in August, with the results published in September 2009. The deadline for product submission will be 27 July. Any developers interested in submitting a product are encouraged to get in touch by emailing martijn.grooten@virusbtn.com.

END NOTES & NEWS

The sixth Conference on Email and Anti-Spam (CEAS) will be held 16–17 July 2009 in Mountain View, CA, USA. For details see http://www.ceas.cc/.

A Mastering Computer Forensics masterclass will take place 22–23 July 2009 in Jakarta, Indonesia. For details see http://www.machtvantage.com/computerforensics.html.

Black Hat USA 2009 will take place 25–30 July 2009 in Las Vegas, NV, USA. Training will take place 25–28 July, with the briefings on 29 and 30 July. For details see http://www.blackhat.com/.

The 18th USENIX Security Symposium will take place 12–14 August 2009 in Montreal, Canada. The 4th USENIX Workshop on Hot Topics in Security (HotSec '09) will be co-located with USENIX Security '09, taking place on 11 August. For more information see http://www.usenix.org/events/sec09/.

The International Cyber Conflict Legal & Policy Conference 2009 will take place 9–10 September 2009 in Tallinn, Estonia. The conference will focus on the legal and policy aspects of cyber conflict. For details see http://www.ccdcoe.org/126.html.

The 7th German Anti-Spam Summit takes place 14–16 September 2009 in Wiesbaden, Germany (the event language will be English). For details see http://www.eco.de/veranstaltungen/7dask.htm.

IMF 2009, the 5th International Conference on IT Security Incident Management & IT Forensics takes place 15–17 September 2009 in Stuttgart, Germany. Experts will present and discuss recent technical and methodical advances in the fields of IT security incident response and management and IT forensics. For more information see http://www.imf-conference.org/.

SOURCE Barcelona will take place 21–22 September 2009 in Barcelona, Spain. The conference will be run in two tracks: Security and Technology, covering security software, application security, secure coding practices, engineering, new tool releases and technology demonstrations; and Business of Security, covering critical decision-making, entrepreneurship, issues of compliance, regulation, privacy laws, disclosure and economics. For full details and registration see http://www.sourceconference.com/.

Hacker Halted 2009 takes place in Miami, FL, USA, 23–24 September 2009. See http://www.hackerhalted.com/.

VB2009 will take place 23–25 September 2009 in Geneva, Switzerland. For the full conference programme including abstracts for all papers and online registration, see http://www.virusbtn.com/conference/vb2009/.

The third APWG eCrime Researchers Summit will be held 13 October 2009 in Tacoma, WA, USA in conjunction with the 2009 APWG General Meeting. eCrime '09 will bring together academic researchers, security practitioners and law enforcement to discuss all aspects of electronic crime and ways to combat it. For more details see http://www.ecrimeresearch.org/.

Malware 2009, the 4th International Conference on Malicious and Unwanted Software, will take place 13–14 October 2009 in Montreal, Quebec, Canada. For more information see http://www.malware2009.org/.

The SecureLondon Workshop on Information Security Audits, Assessments and Compliance will be held on 13 October 2009 in London, UK. See http://www.isc2.org/EventDetails.aspx?id=3812.

RSA Europe will take place 20–22 October 2009 in London, UK. For full details see http://www.rsaconference.com/2009/europe/.

The 17th general meeting of the Messaging Anti-Abuse Working Group (MAAWG) will be held 26–28 October 2009 in Philadelphia, PA, USA. Meetings are open to members and invited participants only. See http://www.maawg.org/.

AVAR2009 will be held 4–6 November 2009 in Kyoto, Japan. For more details see http://www.aavar.org/avar2009/.

SUBSCRIPTION RATES

Subscription price for 1 year (12 issues):

- Single user: $175
- Corporate (turnover < $10 million): $500
- Corporate (turnover < $100 million): $1,000
- Corporate (turnover > $100 million): $2,000
- *Bona fide* charities and educational institutions: $175
- Public libraries and government organizations: $500

Corporate rates include a licence for intranet publication.

See http://www.virusbtn.com/virusbulletin/subscriptions/ for subscription terms and conditions.

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)1865 543153
Email: editorial@virusbtn.com Web: http://www.virusbtn.com/