NOVEMBER 2009

virus BULLETIN

Fighting malware and spam

CONTENTS

ISSN 1749-7027

IN THIS ISSUE

DETECTING BOOTKITS

Alisa Shevchenko and Dmitry Oleksiuk find out whether anti-virus software has learned to cope successfully with Mebroot and MBR infectors in general a few years after the first appearance of this type of malware.
page 11

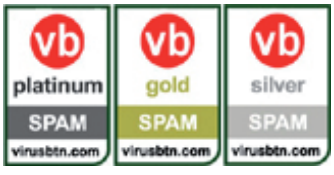
HASH BROWNS

Josh Attenberg and colleagues describe the hashing trick as an effective method for collaborative spam filtering.
page 13

MS SECURITY ESSENTIALS

Microsoft Security Essentials, the long-awaited replacement for the *Windows Live OneCare* package, is finally with us. *VB*'s test team put *Microsoft*'s new free home-user package through its paces, declaring it to be pretty decent overall.
page 18

VBSPAM CERTIFICATIONS



This month's anti-spam comparative review saw yet another increase in the field of competitors with 14 products taking their place on the test bench. Martijn Grooten has the full details.
page 22

'Social engineering threats are a notable concern for mobile device users and are always escalated during the holiday period.'

Ken Dunham, iSIGHT Partners

HAPPY HOLIDAYS: MOBILE MALICIOUSNESS

Since the advent of Timofonica in 2000 there has been a buzz about mobile malicious threats. A boom of mobile malicious code development in 2004 resulted in infections in dozens of countries and thousands of devices. While this was troubling, a more significant and worrying trend driven by financial fraud is now exploiting the mobile device vector.

There has been a rapid surge in the adoption of mobile solutions such as *Blackberry*, *iPhone* and countless other smartphone devices since 2006. Millions of mobile device users rely on their hand-held solutions not only for voice communications but also to perform online banking, surf the Internet, check their email, and more. The reliance on and trust of such devices by the average consumer presents fraudsters with great opportunity.

Starting with more traditional forms of fraud, many 'knock-off' models of mobile devices exist globally, produced and sold in attempts to undercut legitimate market products with cheaper phones which apparently offer increased functionality. This type of brand-based fraud significantly impacts the mobile device market and is difficult for consumers to identify.

Social engineering threats are also a notable concern for mobile device users and are always escalated during the holiday period – targeted attacks are common and are potentially a higher risk at this time of year due to the nature of what and how people communicate with one another at this time. 'Check this out' and 'holiday greetings' are possible spoofed communication vectors for criminals targeting individuals with mobile malicious code. A multitude of ring-tone-based malcode threats will certainly also exist during the 2009 holiday period, impacting both PCs and mobile devices. Old-school social engineering tricks such as the downloading of porn are still in use to trick users into installing mobile device diallers that make outbound calls to premium lines at the expense of the victim. The social engineering vectors are almost limitless, as are the criminals' opportunities for financial fraud.

Mobile device users are now receiving phone calls, SMS messages and emails requesting information about their credit card or other sensitive details. Fraudsters often have all the information they need but a CVV number to perform financial fraud and may engineer a call to a victim to acquire their CVV number. In some advanced cases of social engineering fraudsters have been known to call victims for a one-time password (OTP) value generated from a token used by a victim. If the victim gives out the OTP the fraudsters cash out in real time – often while the victim is still speaking with them on the phone.

Vishing attacks are also on the rise, where VoIP technology is exploited to automate out-of-band broadcast calls to large numbers of mobile devices and/or land lines. The goal is to trick users into entering sensitive details over the phone into an interactive voice-recorded and softphone system on a remote VoIP server. Many consumers don't understand this new type of attack vector and how caller ID can easily be spoofed via VoIP. If reported, these attacks are typically over by the time the authorities attempt to stop and/or investigate them.

As you prepare for the holiday rush, are you planning on purchasing a smartphone device for yourself or as a gift for a loved one? Can you be sure it's a legitimate phone from a trusted brand? After purchasing the device do you know the common best practices for that device to limit the threat vectors? Are you fully aware of the numerous ways that fraudsters will attempt to compromise your device or trick you into revealing sensitive information for financial fraud?

While *VB* readers will understand these threats rather well, most average users of smartphone devices do not and will never understand all of the above (nor want to). The security challenges that lie ahead of our industry are great in light of the challenges identified to date for the mobile market.

NEWS

NIGERIA TAKES STEPS TO CLEAN UP ITS ACT

After years of being inextricably linked to the advance fee fraud scam, the government of Nigeria is launching an offensive to clamp down on the activity. Despite the advance fee fraud scam now being prevalent across the globe, it is widely accepted that the scam originated in Nigeria – indeed, the type of scam is also commonly known as both the Nigerian scam and the 419 scam (419 referring to the section of the Nigerian criminal code violated by the scam). The government now aims to remove Nigeria from the top 10 list of countries with the highest incidence of fraudulent emails.

Nigeria's Economic and Financial Crimes Commission (EFCC) announced last month that, aided by *Microsoft*, it has begun a large-scale crackdown on its indigenous email scammers. More than 800 fraudulent email accounts have already been identified and shut down, while the EFCC anticipates being able to take down 5,000 fraudulent emails per month as well as sending around 230,000 advisory mails to victims and potential victims per month once the operation gathers full pace. So far there have been 18 arrests of individuals suspected of coordinating organized cybercrime rings. The operation, dubbed 'Eagle Claw', is expected to be fully operational within six months.

FACEBOOK WINS AGAINST 'SPAM KING' WALLACE

Facebook has become the latest global giant to be awarded damages in a case against 'Spam King' Sanford Wallace. The social networking company was awarded $711.2 million in damages last month.

The company filed legal action against notorious spammer Wallace after he was found to have hacked into users' accounts, made fake postings and sent fake messages advertising various products and services. A statement on the *Facebook* official blog suggested that the company does not expect to receive 'the vast majority of the award'. Wallace himself failed to appear in court, and the judge referred him to the US Attorney's Office with a request that he be prosecuted for criminal contempt (for which he may face jail time).

Wallace came to prominence as a prolific spammer in the mid 1990s, but in 1998 announced his retirement from the spamming business after facing lawsuits from *AOL* and *CompuServe*. However, he didn't stay off the scene for long – in 2008 *MySpace* was awarded what was at the time a record $230 million in a lawsuit against Wallace and his cohort Walter Rines for spamming and phishing activities. *MySpace* has so far failed to collect its damages from the duo.

Prevalence Table – September 2009

Malware	Type	%
Bredolab	Trojan	35.49%
OnlineGames	Trojan	26.84%
Invoice	Trojan	6.49%
Murlo	Trojan	3.17%
Encrypted/Obfuscated	Misc	3.16%
Downloader-misc	Trojan	3.09%
Mytob	Worm	3.05%
FakeAV	Trojan	2.78%
NetSky	Worm	2.64%
Heuristic/generic	Misc	2.04%
Virut	Virus	1.77%
Agent	Trojan	1.33%
Krap	Trojan	1.27%
Mydoom	Worm	1.08%
Zlob/Tibs	Trojan	0.61%
Lineage/Magania	Trojan	0.56%
Small	Trojan	0.45%
Bagle	Worm	0.39%
Mabezat	Virus	0.33%
Sality	Virus	0.32%
Dropper-misc	Trojan	0.29%
Alman	Worm	0.29%
Basine	Trojan	0.25%
Delf	Trojan	0.21%
Iframe	Exploit	0.19%
Backdoor-misc	Trojan	0.17%
Zbot	Trojan	0.14%
Keylogger-misc	Trojan	0.14%
Mywife/Nyxem	Worm	0.12%
Buzus	Trojan	0.12%
Tiny	Trojan	0.11%
Zafi	Worm	0.09%
Autorun	Worm	0.08%
Others[1]		0.93%
Total		100.00%

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

MALWARE ANALYSIS

PRESCRIPTION MEDICINE

Peter Ferrie
Microsoft, USA

People often ask how we choose the names for viruses. It might seem as if it's in the same way as pharmaceutical companies choose their product names. Zekneol – chemical or virus? In this case, it's a *Windows* virus: W32/Zekneol.

EXCEPTIONAL BEHAVIOUR

After decryption, the virus begins by discarding a number of bytes from the stack. The number of bytes to be discarded is specified in a variable in the virus body. However, the value in this variable is always zero because the polymorphic engine in the virus does not support the generation of fake push instructions.

After 'emptying' the stack, the virus retrieves the return address from it, which points into kernel32.dll. The virus intends to use this as a starting point for a search for the PE header of kernel32.dll. As a precaution, the virus registers a Structured Exception Handler (SEH), which is supposed to intercept any exception that occurs. The virus will search up to 256 pages for the PE header. If the header is not found, then the virus enters an infinite loop. This loop is intentional, it's not a bug. However, if an exception occurs during the search, the handler is reached, along with the first two bugs in the code. After restoring the stack pointer, we see a write to the ExceptionList field in the Thread Environment Block (TEB). Presumably the virus author wanted to unhook the handler, but he forgot to initialize the pointer register first. Thus, the code attempts to write to an essentially 'random' address. This causes a secondary exception, which destroys the handler pointer that is on the stack. What happens next depends on the platform.

On *Windows 2000* and earlier, the damaged handler pointer is assumed to be valid, and so it is used. This of course causes another exception to occur, and the damaged handler pointer is used again, causing yet another exception, and ultimately resulting in an infinite loop. On *Windows XP SP2* and later, the handler pointer is recognized as being invalid, and the application is terminated.

That's the first bug. The second bug occurs on the same instruction. Even if the pointer register were initialized, the wrong value would be written. When registering or unregistering a handler via SEH, the value to write to the ExceptionList field in the TEB is a pointer to a structure. The structure contains a pointer to the handler. The problem is that the virus tries to store the pointer to the handler itself. The reason this happens is that, despite the two values being next to each other on the stack, the virus picked the wrong one.

In fact, there is a third bug in the same region of code. Even if the write succeeds (if the virus initializes the register and chooses the correct pointer to use), the virus attempts to continue the search. The problem is that the search uses several other registers, all of which have been modified as a result of the exception, and none of which are now initialized.

THE PURSUIT OF H-API-NESS

If all goes well, and the virus finds the PE header for kernel32.dll, then the virus resolves some APIs including two which are never used (GetCurrentDirectoryA() and GetWindowsDirectoryA()). The virus uses hashes instead of names, but the hashes are sorted according to the alphabetical order of the string that they represent. This means that the export table needs to be parsed only once for all of the APIs, instead of once for each API, as is common in some other viruses.

After retrieving the API addresses, the virus registers another Structured Exception Handler. The same two bugs exist here regarding the handler behaviour of an uninitialized register and writing the wrong value. The virus uses the same hashing method to resolve an API from user32.dll and several from advapi32.dll (including CryptDecrypt(), which is never used). However, the virus uses the GetProcAddress() API to retrieve the address of the ChecksumMappedFile() API from imagehlp.dll and the SfcIsFileProtected() API from sfc.dll, if those DLLs are available. The use of the GetProcAddress() API avoids a common problem regarding import forwarding. The problem is that while the API name exists in the DLL, the corresponding API address does not. If a resolver is not aware of import forwarding, then it will retrieve the address of a string instead of the address of the code. In this case, support for import forwarding (which the GetProcAddress() API provides) is necessary to retrieve the IsFileProtected() API from sfc.dll, since it is forwarded to sfc_os.dll in *Windows XP* and later.

MISDEEDS AND MISDIRECTION

The virus selects a random number from one to five, which it uses as the number of 'diversion' API calls to make. Then the virus counts the number of 'safe' APIs that it found in the host (this will be described in detail below). The table contains a number of structures, each of which is two DWORDs large. The first DWORD is the RVA of the 'safe' API, and the second one is the number of parameters that the API accepts. However, there is a bug in the counting method. Instead of examining every second DWORD, the virus examines every DWORD for a value of zero. Thus, if an API accepts no parameters, then the parameter count slot will be considered the end of the list. The result is a count

that is both incorrect and invalid, since the end of the list is now misaligned. This bug is essentially harmless, though. The virus chooses randomly from among the APIs, using the wrong number of entries, as calculated previously. However, a more serious bug does exist. The virus multiplies by eight the index of the chosen API. The assumption is that the original count was simply the number of APIs, and therefore multiplying by eight would be the correct behaviour. However, since the count is already too large, and if the table is very full, then as a result of the multiplication the access will be beyond the end of the table. If, for example, it should hit one of the variables that exists after the table, then that variable will be considered the number of parameters to place on the stack. This number might be very large and cause a stack-overflow exception, and a possible hang as described above. Even if the parameter count appeared to be zero, the assumed API itself is still called, which might cause some unexpected behaviour.

If the corresponding slot in the table is empty, then no attempt is made to call the API. If an API does exist, and if that API accepts parameters, then the virus places onto the stack a corresponding number of random values before calling the API. An API is considered 'safe' if it will return an error when passed invalid parameters.

The buggy selection is repeated according to the number of 'diversion' API calls to make, which was chosen previously. The virus then encrypts its memory image, with the exception of a small window, prior to calling the true API. Upon return from the true API, the virus decrypts its memory image, and then performs another set of 'diversion' API calls, as described above. The encryption key for the memory image is changed each time this routine is called. The intention of the routine is to defeat memory scanners that perform their scanning whenever certain APIs are called.

The whole routine, beginning with the first set of 'diversion' API calls, is called whenever the virus wishes to call an API (with two exceptions: CreateThread and GetTickCount are called directly – this is probably an oversight, since nearby APIs within the same routine are called indirectly).

KAMIKAZE CODE

The virus searches within the current directory for '.exe' files whose name begins with 'kaze'. This appears to be a bug, since the first generation version of the virus uses one string, but replicants of the virus use another. However, the string has been duplicated, so the result is always the same.

For each such file, the virus begins by checking if the SfcIsFileProtected API exists. If the API exists, then the virus retrieves the full path of the file, converts the pathname from ASCII to Unicode, and then checks if the file is protected. This is the correct method to determine the protection state. Most viruses that perform the check forget that the API requires the full path to the file. However, if the file is protected, the virus attempts to unmap a view of the file and close some handles. The problem is that the file has not yet been either opened or mapped. Fortunately, the attempt simply returns an error, unless a debugger is present. If a debugger is present, then closing the handle will cause an exception, and a possible hang as described above.

If the file is not protected, then the virus attempts to open it. If the attempt fails, then the virus skips the file, without attempting to unmap or close it. If the open succeeds, the virus maps a view of the file.

The virus contains only one bounds check when parsing the file. That check is simply that the PE header starts within the file. There is no check that the header ends within the file, and the existence of an Import Table is assumed. This means that certain valid but unusual files will cause an exception and a possible hang, as described above. The virus is interested in PE files that are not already infected, and which contain an Import Table that is less than 4,066 bytes large. The virus does not care if the file is really a DLL or a native executable. The infection marker is that the second byte in the time/date stamp in the PE header has a value of 0x36.

The virus places the infection marker immediately, and resizes the file enough to hold the virus code. The virus does not care about any data that has been appended to the file outside of the image. Any such data will be destroyed when the file is resized.

The virus searches for the section with the largest virtual address. For files that run on *Windows NT* and later, this will always be the last section. However, *Windows 9x/Me* files do not have such a requirement. If the virtual size of that section is larger than the physical size, then the virus will not infect the file. However, the infection marker and increased file size remain.

RELOCATION REQUIRED

With a 20% chance, and if the file contains relocations, the virus will relocate the image. The virus parses the relocation table, and applies the relocations to the image using a new image base of 0x10000. After the relocation has been completed, the relocation table is no longer required. There is a bug in the parsing, which is that the virus assumes that the relocation table ends when the page RVA is zero. The assumption is incorrect. The size field in the data directory contains the true size. Further, the virus assumes that any non-zero value is valid, but if the virus is reading data from beyond the end of the relocation table, then it might cause an exception and a possible hang, as described above.

When parsing the relocation data, the virus supports only two types of relocation item. They are the IMAGE_REL_BASED_ABSOLUTE and IMAGE_REL_BASED_HIGHLOW. There are several other documented relocation types, and if one of them is seen, then the virus will hit a breakpoint and possibly hang as described above. However, it is rare for files to use relocation types other than the supported two.

After relocating the image, the virus chooses a new image base randomly. The new image base always points into the upper 2Gb of memory, and is 64kb-aligned. The alignment is a requirement for *Windows NT* and later. It is interesting that the virus appears to have been written to support older versions of *Windows*, since it considers the presence of both imagehlp.dll and sfc.dll to be optional. In fact, imagehlp.dll was introduced in *Windows 98*, and sfc.dll was introduced in *Windows 2000*, so the support goes back a long way. However, the use of 0x10000 as the relocated image base ties the virus to *Windows 2000* and later. The reason for this is that *Windows NT* does not relocate .exe files, and *Windows 9x* and *Me* use 0x400000 as the default image base for relocated files.

DEP-RECATED CODE

The virus increases the size of the last section by 139,264 bytes, and changes the section attributes to read/write/initialized. Unfortunately for the virus author, the executable attribute is not set explicitly. As a result, if the attribute was not already set in the original file, then the virus will fail to execute on systems which have Data Execution Protection enabled.

The virus saves information about the address and size of resources and imports. The virus pays special attention to the imports, parsing the table and saving pointers and ranges. However, as before, there is no bounds checking while saving the values, so a very large Import Table could cause corruption of other entries in the list.

The virus will now choose a decryptor method to use. The virus uses a crypto-based method 80% of the time. For the other 20% of the time, it uses a simple 32-bit add-based decryptor.

CRYPTONITE

If the crypto-based decryptor is chosen, the virus copies the host's Import Table to the original end of the last section and updates its RVA in the data directory. The size of the Import Table is increased by the size of one Import Table record, and the Bound Import Table data directory entry is erased. The virus appends to the Import Table an entry that refers to the advapi32.dll file. The 'advapi32.dll' string is appended to the section, at a random location beyond the end of the Import Table. The five crypto-related APIs that the virus uses (CryptAcquireContextA, CryptCreateHash, CryptHashData, CryptDeriveKey and CryptDecrypt) are appended to the Import Table, interspersed with one to four imports chosen randomly from a set of 75 'safe' APIs from the advapi32.dll file. The name of each API is placed at a random location beyond the end of the Import Table. The virus also contains code to replace the unused bytes with random data, but this routine is never called.

SAFETY IN NUMBERS

The virus examines the host Import Table for references to DLLs that it knows contain 'safe' APIs. Those DLLs are kernel32.dll, ws2_32.dll, user32.dll and gdi32.dll. If one of the 'safe' DLLs is imported, then the virus searches for references to the 'safe' APIs. If any 'safe' API is imported, then the virus adds the reference to a table within the virus body. There is what might be considered a bug in the search routine. The virus searches the entire Import Table for a reference to the first 'safe' DLL, then searches for references to the 'safe' APIs of that DLL, then searches for a reference to the second 'safe' DLL, and so on. However, if the host does not import anything from one of the 'safe' DLLs, then the virus stops searching completely. None of the following DLLs will be checked, and no more 'safe' APIs will be added to the table. Thus, in the extreme case, if the host does not import anything from kernel32.dll, then no 'safe' APIs will be added at all.

The virus then copies the decryptor, and optionally inserts calls to the 'safe' API if the crypto-based method was chosen. As before, the method to choose from the 'safe' API table uses a count that is too large, resulting in empty slots being seen, and thus no API call being inserted in that instance. However, there are multiple places within the decryptor where APIs can be inserted, which increases the chance that at least one of them will succeed.

BAIT AND SWITCH

If the crypto-based method was chosen, the virus changes the attributes of each section to read/write/initialized, until the section containing the entrypoint is seen. However, the virus chooses random locations only from within the section that contains the entrypoint. The virus saves the contents from each of the locations that were chosen, since they will be replaced later by parts of the decryptor.

The virus then constructs a new decryptor. The decryptor is described using a p-code language, which gives it great flexibility. The p-code contains only 57 instructions, but they are quite capable of producing a seemingly wide

variety of code. However, the characteristics of that code are instantly recognizable, and the instruction set used is very small. Some of the instructions are also called recursively, so that, for example, a simple register assignment first becomes a series of assignments and adjustments through other registers. The two types of decryptor together use fewer than half of the possible instructions, but internally those instructions use all but one of the remaining instructions (the missing one is a 'test' instruction involving a memory address and a constant). While interpreting the p-code, the virus resolves the API calls, both real and fake, and inserts random numbers for the parameters to the fake APIs, and real parameters for the real APIs.

If the virus has relocated the image, then it will also encrypt some of the blocks by using relocation items (for a description of the process, see *VB*, April 2001, p.8). The virus creates a new relocation table that contains only the items for the decryptor, by overwriting the original relocation table in the host. However, in contrast to ordinary files, the virus places the relocation items in decreasing order in the file, and calculates some page addresses using values that are not page-aligned. These two characteristics immediately make those files suspicious. The virus stops applying relocations when fewer than 328 bytes of space remain in the original table. There is a bug here, though, which is that if the original table was less than 328 bytes long, then the virus sets the table size to zero bytes. The resulting file will no longer load, because when an image must be relocated, *Windows* requires that a relocation table contains at least the page address and the number of relocation items (even if the number of items is zero).

ROCK CITY FUNK

At this point, the virus copies itself to the file in unencrypted form. The encryption is performed next, on the copy of the virus body, using the chosen method. The crypto-based method uses a 128-bit RC4 cipher, with an MD5 hash as the key. The key is derived from the four-byte Import Lookup Table RVA in the first entry of the new Import Table.

The virus increases the size of the image by 139,264 bytes, and if the ChecksumMappedFile API is available, then the virus uses it to calculate a checksum for the file. This results in a file having a checksum that might not have existed before. Finally, the file is unmapped and closed. The virus then searches for the next file to infect.

Once all files have been examined, the virus displays the message 'Infecté', if not running a first generation of the code. If the executing image is not a first generation of the code, then the virus changes the section attributes to read/write/executable for the section that contains each block. Of course, it's too late to save the virus on DEP-enabled systems. Since all of the blocks are chosen from the same section that contains the entrypoint, changing the attributes multiple times is ultimately pointless. It appears that the virus author wanted to support blocks in multiple sections, but this virus does not support it. After changing the attributes of the blocks, the virus restores their contents.

GOSSAMER THREADS

After restoring the host to an executable state, the virus creates a thread to search drives for other files, then run the host. The thread registers another Structured Exception Handler. However, this time only the second bug is present. The virus initializes the pointer correctly, but the value to write is still wrong. Further, if an exception occurs, then the virus wants to exit the thread, but the problem is that the code uses another register which has been modified as a result of the exception, and is now not initialized.

If no exception occurred, then the virus begins with drive 'B:' and proceeds though drive letters until the first drive is found which is either fixed or removable. Only that drive will be examined. This might also be considered a bug, but the loop contains no other exit condition, so perhaps it was intentional to stop after one drive. The idea of starting with drive 'B:' rather than 'A:' could also introduce a bug, in the (admittedly rather unlikely) event that the only drive on the system is 'A:'. In that case, all possible values would be tested, but even so, eventually the value would wrap around and the 'A:' drive would be found. When an appropriate drive is found, the virus sleeps for one second before beginning the search for files. The search is for 'kaze' files, as described above. Upon completing the search for files, the virus will search for directories. If a directory is found, then the virus will enter the directory and begin the search again, after sleeping for one second, as before. If no other directories are found, then the virus will step out of the current directory and resume the search. After all directories have been examined, the thread will exit.

In the event that the host process finishes before the virus thread exits, the virus thread will be forcibly terminated. This could result in a corrupted file if the virus was in the act of infecting it at the time.

CONCLUSION

Zekneol certainly appears to be a complicated virus, but looks can be deceiving. The crypto-based decryptor has so many tell-tale signs that detection is straightforward; the simple decryptor is really very simple; and the new relocation table looks like no other.

As for how we choose the names for viruses, that's a question for another day.

FEATURE 1

DATA TAINTING FOR MALWARE ANALYSIS – PART TWO

Florent Marceau
CERT-LEXSI, France

In this three-part series Florent Marceau studies the use and advantages of full virtualization in the security field. Following an introduction to full virtualization in part one (see *VB*, September 2009, p.6), this part looks at the limitations of the technology.

FULL VIRTUALIZATION

Many previous studies have been published on the subject of full virtualization and its limitations. In particular, virtualization detection has been the subject of many publications [1]. For semi-assisted virtualization with a hypervisor, detection is mainly focused on detection of the hypervisor itself by looking for the relocation of key structures of the operating system, such as the IDT (Interrupt Descriptor Table), the GDT (Global Descriptor Table), etc. (c.f. [2]).

The main and inherent drawback of full virtualization is the need to thoroughly implement all the characteristics of the architecture. This implementation is sometimes incorrect or incomplete, and then becomes detectable. This kind of problem can be exploited to develop detection codes for targeted virtual machines [3, 4] simply through the use of instruction sequences that will react differently on a real CPU and on an emulated CPU. For example, you can use the FPU (Floating Point Unit) mnemonic 'fnstenv' to push the FPU environment into memory, and then use the FPUInstructionPointer field of this structure as the address of the last FPU opcode. A real CPU will respond normally, but on an emulated CPU such as a *Qemu*, this generally unused field is never updated. It is thus possible to detect the presence of the emulator.

A much simpler but very popular method of detecting virtualization is simply to check for the names of the manufacturers of the different devices. The *Qemu* hard drive, for example, is named 'QEMU HARDDISK' by default – this easily reveals the emulator's presence (and is also easy to remedy). Moreover, some detection kits have been seen shared in the malware community. These will compare the serial number of the *Windows* host with the serial number of some well-known public sandboxes, such as *Anubis* or *CWSandbox*. Another very common method is to monitor the potential activity of an active user (mouse movement or other). Finally, one common countermeasure – targeting all kinds of automated analysis platforms – is to let the code sleep for a long period (an average of 20 minutes) to break out of the analysis time frame.

TAINTED TAGS PROPAGATION POLICY

Data tainting is a mechanism that allows us to track the full propagation of a given set of data on an information system. A full description was given in part one of this series (see *VB*, September 2009, p.6).

We must now define a tainted tags propagation policy. This policy is directly dependent on the potential processing that could be applied to the data we want to track. We must apply our chosen policy by modifying each opcode handler on the virtual CPU.

There are two major considerations in the definition of this policy, which are the type of data to track (e.g. code, confidential data or other data) and the potential processing the data may go through (e.g. obfuscation, encryption).

Given the context and previous considerations, the heavy transformations imposed on our data through obfuscation and encryption could result in the loss of the tainted marks of legitimately marked data. It would therefore be logical to apply a more persistent propagation strategy to preserve the maximum number of tainted tags. Unfortunately, a propagation policy that is too permissive will create illegitimate tainted data, generating taintmap pollution which leads to many false positives. This pollution will consult a lot of binary data, strictly speaking 'viral', that will massively pollute the output and drown our configuration file in garbage, making its use almost impossible. Pollution will also be caused by legitimate data owned by the exploitation system that has been merged with viral data. Here, we have to find a compromise. Later, we'll establish the configuration file characterization, but for the moment we are just searching to keep a consistent propagation on the viral binary.

Many previous studies have been published on the analysis of execution and data flow, sometimes using static analysis, sometimes dynamic, and sometimes both. Some of these studies are applicable only to high-level languages while others can be applied to closed-source binary. In our case we are limited to dynamic analysis for closed-source binary.

Let's examine an overview of the theory.

A perfect propagation is exclusive; this means that strictly all data to be monitored is marked as tainted and none other. How do we define 'monitored' data?

A lot of studies of data tainting consider the problem from a confidentiality point of view: private data is marked as tainted and the propagation mechanism is used to ensure that this data can't illegitimately leave the SI (information

system). From this point of view, we want to assure the SI security integrity and, as defined in the non-interference model created by Goguen and Meseguer [5] (dealing with both users and data), the data must under no circumstances interfere with other data that is not explicitly allowed by the SI security policy.

Although our context requires exactly this kind of non-interference between the data to be monitored and the other data, the type of data we wish to track is radically different: our data is composed of the malicious software data and executable code (and there is no requirement for confidentiality). If in both cases the propagation should ideally be applied without loss, our tainted data has a much larger surface of interaction with the exploitation system, especially because the code will be executed with administrator privileges, allowing partial or complete corruption of the operating system by the malware. Thus, our monitored data clearly breaks out of the established security policies. Worse, since some of our tracked data are pieces of code, the attackers have a lot of leeway to perform various emulation detections such as those described earlier, and also to implement anti-tainting techniques via different covert channels (which we will discuss briefly later).

We assume here that the tainting propagation is only applied on particular mnemonic types like the assignment, arithmetic and the logical operation types. This is called 'explicit direct flow' tracking. It's the classical dynamic data tainting implementation. We'll see that in some cases it will not be enough to keep a consistent propagation.

For example, a difficult scenario would be to handle the use of a tainted value as an index in a non-tainted character array (since the array was originally on the system and therefore not derived from monitored code). The generated strings are then probably interesting but require the implementation of tainting propagation between the pointer and the pointed value. This is perfectly achievable. However, in the case of an object code that will naturally make extremely intensive use of pointers, this will lead to massive propagation pollution.

Let's look at another illustration of this kind of blind view. A doubleword (32 bits) received from the network is an array of bit flags used by our process. Bit 7 of these flags will be controlled as follows:

```
(1) mov eax,[our_data]
(2) and eax,0x80
( ) je skip
(3) mov ebx,0xffffffff
( ) skip:
```

In step (1), eax contains our value and will be tainted. In step (2), eax would contain only the bit 7 to control. In a permissive policy eax remains tainted. This bit, which is considered a part of the data to track, is controlled and will influence the conditional jump. Thus, it defines the eventual assignment of the register ebx. Should ebx be tainted? If so, this would require propagation at the eflag bits level in order to apply propagation on the conditional underlying code block. Strictly speaking, we should do it, since the ebx value is derived directly from a tainted value, but in this case it would introduce binary pollution ({0xffffffff}). This is a difficult choice.

The previous case is known as indirect dependence (or control dependence) on our explicit flows. Another way to manage these cases without having to propagate at an eflag bits level is to taint the PC (Program Counter, EIP on x86 architecture) at each comparison (or on a mnemonic that will affect eflag) by the tainted tag value of the operand involved. Therefore, after each conditional jump, if the Program Counter is tainted, the different values assigned in the underlying basic block (in the static analysis meaning) will be tainted as well. The Program Counter taint value will then change at the end of the basic block or on a new comparison.

Note that if this method addresses the previous problem, it will be efficient only with a very simple control flow. It is extremely easy to modify the previous example in order to evade this implementation of control dependency tracking:

```
( ) mov eax,[our_data]
( ) and eax,0x80
( ) je skip
( ) xor ecx,ecx
(1) cmp ecx,1
( ) je skip2
( ) nop
( ) skip2:
( ) mov ebx,0xffffffff
( ) skip:
```

We simply have to force a new condition on dummy untainted values in step (1) in order to remove the tainting mark of the Program Counter, and consequently we lose the tainting on the ebx register. To address this, we could consider a stacking of the different taint values of the Program Counter in function of the flow control call depth. Obviously, this is completely inefficient with obfuscated code; indeed the purpose of the packer is precisely to add many opaque predicates that will add complexity to the control flow graph. This, combined with a technique of hashing the legitimate control flow [6], will finally make our previous implementation obsolete and possibly vulnerable. Moreover, in our context, the initial state involves a large volume of legitimately tainted data (between 15ko and 1Mo on average). The tracking of indirect dependencies (such as

for pointer dependencies) will generate too great a degree of pollution.

Despite all these problems, we have an interesting advantage given the fact that we mark the monitored code; we can consider implementing the propagation of control dependencies (and pointers) only from a tainted piece of code. The inner workings of *Qemu* could lend itself quite well to this modification, in the sense that *Qemu* itself uses basic blocks (this could be developed further in the future).

However, the tracking of indirect dependencies doesn't guarantee that no legitimate tainted marks will be lost. The problem lies in tracking implicit indirect flows (a set of assignments brought about by the non-execution of a piece of code). With a similar form to the previous example:

```
( ) mov byte prt al,[our_data]
( ) mov ecx,0xff
( ) do_it:
(3) mov ebx,1
(1) cmp al,cl
( ) je skip
(2) xor ebx,ebx
( ) skip:
(4) test ebx,ebx
( ) jne done
( ) loop do_it
( ) done:
```

In this new case (example taken from [7]), we loop on ecx, while the value of cl is different from the value in al that is tainted (1), the register ebx is then tainted (2), and at each new iteration the taint of ebx is deleted by the assignment (3). When equality between al and cl is attained in (1), the value of ebx remains unchanged. Ebx is then not tainted when in (4) it validates its non null value as a loop release condition. We then reach the label 'done' with a cl value equal to our tainted value, but without being able to propagate this taint mark on the register ecx.

There are various methods to propagate the tainting despite this kind of implicit indirect flow, some of them use static pre-analysis, others only apply on theoretical machine models dedicated to research (c.f. [8]). There is no absolute solution here.

Another common problem referred to in the literature dealing with the data flow is the use of covert channels, but in our context we are not dealing with privacy, and information leaks through time covert channels, as discussed in [7] don't affect us. However, other covert channels could. The previous example (pointer propagation) which uses untainted data that was originally present on the system to illegitimately generate untainted viral data is proof of this. And that's not the only example. Let's consider malicious software using a configuration file consisting of only a cryptographic (hash) of its target names strings. It would then read the current navigation site name, generate a (hash) digest and compare it with those in its configuration file. These types of blind views leave the solution completely ineffective.

CONCLUSION

Data tainting is a powerful tool but very difficult to calibrate. The main difficulty lies in establishing a propagation policy that is sufficiently delicate, but that will not involve full pollution of the taintmap, and then in its implementation. This can be done over time by calibrating against different samples of malicious software.

In the third part of this article we will look at the implementation of data tainting.

REFERENCES

[1] Raffetseder, T.; Kruegel, C.; Kirda E. Detecting System Emulators. http://www.seclab.tuwien.ac.at/papers/detection.pdf.

[2] Rutkowska, J. Red Pill... or how to detect VMM using (almost) one CPU instruction. http://www.invisiblethings.org/papers/redpill.html.

[3] Ferrie, P. Attacks on More Virtual Machine Emulators. http://pferrie.tripod.com/papers/attacks2.pdf.

[4] Ormandy, T. An Empirical Study into the Security Exposure to Hosts ofHostile Virtualized Environments. http://taviso.decsystem.org/virtsec.pdf, http://www.iseclab.org/papers/ttanalyze.pdf.

[5] Goguen, J.A.; Meseguer, J. Security Policy and Security Models, Proc. IEEE Symp. Security and Privacy, pp.11–20, 1982. http://www.cs.ucsb.edu/~kemm/courses/cs177/noninter.pdf.

[6] Collberg, C.; Thomborson, C.; Low D. A Taxonomy of Obfuscating Transformations. http://www.cs.arizona.edu/~collberg/Research/Publications/CollbergThomborsonLow97a/A4.pdf.

[7] Cavallaro, L.; Saxena, P.; Sekar R. Anti-Taint-Analysis: Practical Evasion Techniques Against Information Flow Based Malware Defense. http:// www.seclab.cs.sunysb.edu/seclab/pubs/ata07.pdf.

[8] Le Guernic, G. Confidentiality Enforcement Using Dynamic Information Flow Analyses. http://tel.archives-ouvertes.fr/docs/00/19/86/21/PDF/thesis_report.pdf.

FEATURE 2

DETECTING BOOTKITS

Alisa Shevchenko, Dmitry Oleksiuk
eSage Lab, Russia

This is a short essay about the generic detection of MBR-infecting malware and, in a wider sense, the generic detection of malware.

INTRODUCTION

As a previous tool we developed – TDSS remover (see *VB*, August 2009, p.6) – proved to be popular with users, we decided to continue exploring the capabilities and attempting to fix the shortcomings of anti-virus software.

A concept presented at a recent conference – the Stoned Bootkit – reminded us of another popular and poorly managed threat: the Mebroot (aka Sinowal or Torpig) trojan, and MBR infectors in general. So we decided to find out whether, a few years after the first appearance of this type of malware, anti-virus software has learned to cope successfully with it.

A very simple test was performed in order to exercise the capabilities of different software in detecting and removing MBR-infecting malware, as well as to explore the software's approaches to such detection. Despite the test's relatively amateur methodology, the results clearly showed that most anti-virus software is far from able to cope successfully with MBR-infecting malware. It also showed that most anti-virus software detects MBR-infecting malware by signature matching, which means that any Mebroot specimen can be made undetectable in a matter of minutes.

We decided to create a trivial tool presenting a generic approach to the detection and cleaning of MBR-infecting malware.

BACKGROUND

Mebroot's boot-code-infecting capability is based entirely on the *eEye* Boot Root concept [1] presented at Black Hat 2005. Beyond the concept, Mebroot variants have driver-loading and self-hiding functionality, the latter of which makes the trojan's detection and removal particularly tricky.

Let us remind you about Mebroot's basic features:

1. Mebroot starts from a modified piece of the Master Boot Record code. It doesn't have its own executable file on the filesystem; instead, it stores its code in the MBR and first disk sectors.
2. During system boot, malicious boot code hooks IoInitSystem after the operating system kernel code is read from disk.
3. The IoInitSystem injection provides mapping of a malicious driver into kernel memory.
4. The malicious driver code hooks filesystem drivers, so that an attempt at reading the system MBR would return a seemingly normal boot code.
5. Finally, payload code is injected into user-mode processes from the driver.

THE STONED BOOTKIT

Technologically, the Stoned Bootkit [2] is no different from Mebroot where MBR infection is concerned. This is exactly why it is frustrating that anti-virus tools fail to detect it.

THE TEST

The main objective of the test was to figure out whether anti-virus tools can detect and remove MBR malware in general, rather than just known Mebroot variants. The idea is that those that can, would certainly succeed in the detection of a theoretical new Mebroot variant which is different from an ordinary Mebroot only in its boot code.

To emulate such a piece of malware, a regular Mebroot body (MD5: c8b9853a2a40ab6e9f0363397386f86e [3]) was utilized. We applied a simple obfuscation to the real Mebroot's boot code, so that it could no longer be detected by signature.

Two other test goats were:

- A regular, second-generation Mebroot variant (same MD5) – as a historical, 'must succeed' case.
- The above-mentioned Stoned Bootkit – as a real-world 'new challenge'.

We focused on testing specific anti-Mebroot tools, since they must embody anti-virus best practices. Some other cleaning tools and anti-virus solutions were also tested. In the results table, target software is grouped as follows:

1. A random selection of major anti-virus solutions.
2. Specific anti-Mebroot anti-virus tools.
3. Non-specific advanced cleaning tools from anti-virus vendors.
4. A third-party anti-rootkit solution.

The test conditions were kept simple:

1. All tests were run on the same snapshot of *VMWare*, i.e. in identical conditions.
2. *Windows 2003 Server* was installed on *VMWare*.
3. The latest stable releases of software were installed.

	Product name	Version	Sinowal-b	Sinowal-b modified	Stoned Bootkit
1	McAfee VirusScan	13.15.101	+/+	–/–	–/–
	Kaspersky Antivirus 2010	9.0.0.463	+/+	+/–	–/–
	ESET NOD32 Antivirus	4.0.437.0	+/–	+/–	+/–
	avast! Professional Edition	4.8.1356.0	–		
2	ESET Mebroot Remover	1.7	+/+	+/–	–/–
	Norman Sinowal Cleaner	2008/05/13	–		
	Symantec Trojan.Mebroot Removal Tool	1.0.1.0	–		
3	Dr.Web CureIt!	5.0.2.9230	+/+	–/–	–/–
	F-Secure BlackLight	2.2.1092.0	–		
	Avast! Virus Cleaner	1.0.211	–		
4	RootRepeal	1.3.5.0	+/+	+/+	–/–

Table 1: Test results for detection and disinfection of three pieces of malware ('+' signifies the product detects/disinfects successfully, '–' signifies the product fails to detect/disinfect successfully).

4. Anti-virus software was configured to provide maximum protection.
5. Anti-virus databases were up to date.

The test results can be seen in Table 1.

ANALYSING THE RESULTS

As can be seen from the results table, none of the anti-virus solutions tested is ready for a simple new Mebroot.

Q: Is it easy to produce a new Mebroot variant that would be undetected by the listed software?

A: It is as trivial as a 10-minute exercise in assembly.

Q: Why is *ESET Antivirus* the only software to detect the Stoned Bootkit?

A: Probably because *ESET* is the only anti-virus among those listed that adds signature detections for proof-of-concept code.

Q: Why did *ESET Antivirus* fail to cure a regular Mebroot infection in the first test, while the *ESET Mebroot Remover* tool succeeded in the same task?

A: Actually, *ESET Antivirus* does cure the Mebroot infection. But, because such cleaning requires non-trivial scripting manipulations, we decided to put a '-' in the results table.

Q: Why did some specific anti-Mebroot tools and some advanced virus cleaning tools fail completely?

A: As opposed to automatically updated anti-virus solutions, stand-alone tools are not updated regularly, and thus easily and quickly become outdated. This is not a problem unless a stand-alone tool relies on signatures or other fast-expiring technology, while its nature is to rely on advanced generic solutions.

Q: In the second test, why did most of the software succeed in detecting an active rootkit, but fail to disinfect it?

A: Probably because they detected (and tried to cure) a Mebroot driver in memory while ignoring (and thus missing the fix of) the unknown boot code.

Q: Why did software ignore the modified Mebroot boot code?

A: Probably because a boot code detection is triggered by a known signature and not triggered by modified boot code. Even stand-alone, non-standard boot code is worthy of suspicion. In combination with invisibility, it presents clear evidence of an MBR infector.

Q: Why did *RootRepeal* succeed in the first two tests, and fail in the last?

A: It looks like *RootRepeal* is the only software to implement the anomaly-based detection of MBR malware mentioned in the previous paragraph. A detection is triggered if a custom boot code is found, *and* if it is hidden. In this case, the boot sector is disinfected. Stoned Bootkit isn't detected since it doesn't hide.

Q: What is the idea behind detecting MBR-infecting malware generically?

A: A generic detection is the detection of the essential characteristics of a malware family. As an example, the essential characteristic of any Mebroot-like malware is boot

code infection. Thus, a generic detection of Mebroot-like malware would be detecting boot code anomalies. With such an approach, detection and disinfection of the driver in memory and other malware evidence can be skipped, because cleaning of the boot code will cure an MBR infector completely.

Q: Why is generic detection necessary?

A: Because a detection that can be bypassed in 10 minutes is a waste.

BOOTKIT REMOVER

We created a simple tool that is capable of detecting and disinfecting MBR malware: Bootkit Remover [4].

In the tool's output, three verdicts are possible:

1. Boot code is clean
2. Boot code is modified
3. Boot code is hidden by a rootkit.

Modified boot code can be cleaned by launching the tool with the 'fix' command. In this case, the infected MBR will be overwritten by the operating system's default boot code. Without an infected boot code the Mebroot (or similar malware) will fail to start at the next reboot, so no further cleaning is necessary.

Currently the tool does not recognize custom boot sector code (such as GRUB or Lilo), which means that the second verdict ('boot code is modified') will not necessarily reflect a malicious boot code modification. However, all MBR malware hides its boot code, which means that in case of an MBR infection one will always get the third verdict.

It should be underlined that we are not claiming to present an infallible technology. Basically, Bootkit Remover is an advanced analogue of fixmbr with rootkit detecting capabilities. At the same time, the tool does allow easy detection and disinfection of virtually any piece of MBR malware, thus demonstrating the concept of generic detection of the latter.

REFERENCES

[1] http://www.blackhat.com/presentations/bh-usa-05/bh-us-05-soeder.pdf.

[2] http://www.stoned-vienna.com/.

[3] http://www.virustotal.com/analisis/b29a3d803c513b4ce3b5e10c1455669ccc3581b3d01270840d509af70e3b4130-1254266311.

[4] http://esagelab.com/resources.php?s=bootkit_remover.

FEATURE 3

COLLABORATIVE SPAM FILTERING WITH THE HASHING TRICK

Josh Attenberg
Polytechnic Institute of NYU, USA

Kilian Weinberger, Alex Smola, Anirban Dasgupta, Martin Zinkevich
Yahoo! Research, USA

User feedback is vital to the quality of the collaborative spam filters frequently used in open membership email systems such as *Yahoo! Mail* or *Gmail*. Users occasionally designate emails as spam or non-spam (often termed as ham), and these labels are subsequently used to train the spam filter. Although the majority of users provide very little data, as a collective the amount of training data is very large (many millions of emails per day). Unfortunately, there is substantial deviation in users' notions of what constitutes spam and ham. Additionally, the open membership policy of these systems makes it vulnerable to users with malicious intent – spammers who wish to see their emails accepted by any spam filtration system can create accounts and use these to give malicious feedback to 'train' the spam filter in giving their emails a free pass. When combined, these realities make it extremely difficult to assemble a single, global spam classifier.

The aforementioned problems could be avoided entirely if we could create a completely separate classifier for each user based solely on that user's feedback. Unfortunately, few users provide the magnitude of feedback required for this approach (many not providing any feedback at all). The number of emails labelled by an individual user approximates a power law distribution. Purely individualized classifiers offer the possibility of excellent performance to a few users with many labelled emails, at the expense of the great many users whose classifiers will become unreliable due to a lack of training data.

This article illustrates a simple and effective technique that is able to balance the wide coverage provided by a global spam filter with the flexibility provided by personalized filters. By using ideas from multi-task learning, we build a hybrid method that combines both global and personalized filters. By training both the collection of personal and global classifiers simultaneously we are able to accommodate the idiosyncrasies of each user, as well as provide a global classifier for users that label few emails. In fact, as is well known in multi-task learning [1], in addition to improving the experience of users who label many examples, this multi-task learning approach actually mitigates the impact

of malicious users on the global filter. By offering specific consideration to the intents of the most active and unusual users, a global classifier is created that focuses on the truly common aspects of the classification problem. The end result is improved classifier performance for everyone, including users who label relatively few emails.

With large-scale open membership email systems such as *Yahoo! Mail*, one of the main hurdles to a hybrid personal/global spam filter is the enormous amount of memory required to store individual classifiers for every user. We circumvent this obstacle with the use of the hashing trick [2, 3]. The hashing trick allows a fixed amount of memory to store all of the parameters for all the personal classifiers and a global classifier by mapping all personal and global features into a single low-dimensional feature space, in a way which bounds the required memory independently of the input. In this space, a single parameter vector, w, is trained which captures both global spam activity and the individual aspects of all active users. Feature hashing provides an extremely simple means of dimensionality reduction, eliminating the large word-to-dimension dictionary data structure typically needed for text-based classification, providing substantial savings in both complexity and available system memory.

1. HASHING TRICK

The standard way to represent instances (i.e. emails) in text classification is the so-called bag-of-words approach. This method assumes the existence of a dictionary that contains all possible words and represents an email as a very large vector, $\vec{x}$, with as many entries as there are words in the dictionary. For a specific email, the i^{th} entry in the vector contains the number of occurrences of word i in the email. Naturally, this method lends itself to very sparse representations of instances and examples – as the great majority of words do not appear in any specific text, almost all entries in the data vectors are zero. However, when building a classifier one often has to maintain information on all words in the entire corpus (e.g. in a weight vector), and this can become unmanageable in large corpora.

The hashing trick is a dimensionality reduction technique used to give traditional learning algorithms a foothold in high dimensional input spaces (i.e. in settings with large dictionaries), by reducing the memory footprint of learning, and reducing the influence of noisy features.

The main idea behind the hashing trick is simple and intuitive: instead of generating bag-of-word feature vectors through a dictionary that maps tokens to word indices, one uses a hash function that hashes words directly into a feature vector of size b. The hash function $h : \{Strings\} \rightarrow [1..b]$ operates directly on strings and should be approximately uniform[1].

In [3] we propose using a second independent hash function $\xi : \{Strings\} \rightarrow \{-1, 1\}$, that determines whether the particular hashed dimension of a token should be incremented or decremented. This causes the hashed feature vectors to be unbiased, since the expectation of the noise for any entry is zero. The algorithm below shows a pseudo-code implementation of the hashing trick that generates a hashed bag-of-words feature vector for an email:

```
hashingtrick([string] email)
  x⃗ = 0⃗
  for word in email do
    i = h(word)
    x⃗_i = x⃗_i + ξ(word)
  end for
return x⃗
```

The key point behind this hashing is that every hashed feature effectively represents an infinite number of unhashed features. It is the mathematical equivalent of a group of homographs (e.g. lie and lie) or homophones (e.g. you're and your) – words with different meanings that look or sound alike. It is important to realize that having two meanings of the same feature is no more and no less of an issue than a homograph or homophone: if a computer can guess the meaning of the feature, or more importantly, the impact of the feature on the label of the message, it will change its decision based upon the feature. If not, then it will try to make its decision based on the rest of the email. The wonderful thing about hashing is that instead of trying to cram a lot of different conflicting meanings into short words as humans do, we are trying to randomly spread the meanings evenly into over a million different features in our hashed language. So, although a word like 'antidisestablishmentarianism' might accidentally run into 'the', our hashing function is a lot less likely to make two meaningful words homographs in our hashed language than those already put there by human beings.

Of course there are so many features in our hashed language, that in the context of spam detection most features won't mean anything at all.

In the context of email spam filtering, the hashing trick by itself has several great advantages over the traditional dictionary-based bag-of-words method: 1. It considers even low-frequency tokens that might traditionally be ignored to keep the dictionary manageable – this is especially useful in view of attacks by spammers using rare variants of words

[1] For the experiments in this paper we used a public domain implementation of a hash function from http://burtleburtle.net/bob/hash/doobs.html.

(e.g. misspellings like 'viogra'). 2. Hashing the terms makes a classifier agnostic to changes in the set of terms used, and if the spam classifier is used in an online setting, the hashing trick equips the classifier with a dictionary of effectively infinite size, which helps it adapt naturally to changes in the language of spam and ham. 3. By associating many raw words in its 'infinite' dictionary (most of which never occur) with a single parameter, the meaning of this parameter changes depending upon which words are common, rare, or absent from the corpus.

So, how large a language can this hashing trick handle? As we show in the next section, if it allows us to 'square' the number of unhashed features we may possibly see, it could help us handle personalization.

2. PERSONALIZATION

As the hashing trick frees up a lot of memory, the number of parameters a spam classifier can manage increases. In fact, we can train multiple classifiers which 'share' the same parameter space [3]. For a set of users, U, and a dictionary size d, our goal is to train one global classifier, $\vec{w}_0$, that is shared amongst all users and one local classifier, $\vec{w}_u$, for each user $u \in U$. In a system with $|U|$ users, we need $|U| + 1$ classifiers. When an email arrives, it is classified by the combination of the recipient's local classifier and the global classifier $\vec{w}_0 + \vec{w}_u$ – we call this the hybrid classifier. Traditionally, this goal would be very hard to achieve, as each classifier $\vec{w}_u$ has d parameters, and hence the total number of parameters we need to store becomes $(|U| + 1)d$. Systems like *Yahoo! Mail* handle billions of emails for hundreds of millions of users per day. With millions of users and millions of words, storing all vectors would require hundreds of terabytes of parameters. Further, to load the appropriate classifier for any given user in time when an email arrives would be prohibitively expensive.

The hashing trick provides a convenient solution to the aforementioned complexity, allowing us to perform personalized and global spam filtration in a single hashed bag-of-words representation. Instead of training $|U| + 1$ classifiers, we train a single classifier with a very large feature space. For each email, we create a personalized bag of words by concatenating the recipient's user id to each word of the email[2], and add to this the traditional global bag of words. All the elements in these bags are hashed into one of b buckets to form a b-dimensional representation of the email, which is then fed into the classifier. Effectively, this process allows $|U| + 1$ classifiers to share a b-dimensional parameter space nicely [3]. It is important to point out that the one classifier $\vec{x}$ – over b hashed features – is trained *after* hashing. Because b will be much smaller than $d \times |U|$, there will be many hash collisions.

However, because of the sparsity and high redundancy of each email, we can show that the theoretical number of possible collisions does not really matter for most of the emails.

Moreover, because the classifier is aware of any collisions before the weights are learned, the classifier is not likely to put weights of high magnitude on features with an ambiguous meaning.

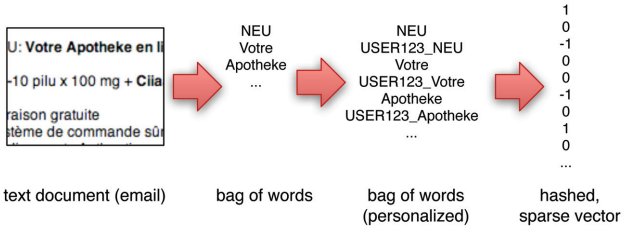


Figure 1: Global/personal hybrid spam filtering with feature hashing.

Intuitively, the weights on the individualized tokens (i.e. those that are concatenated with the recipient's id) indicate the personal eccentricities of the particular users. Imagine for example that user 'barney' likes emails containing the word 'viagra', whereas the majority of users do not. The personalized hashing trick will learn that 'viagra' itself is a spam indicative word, whereas 'BARNEY_viagra' is not. The entire process is illustrated in Figure 1. See the algorithm below for details on a pseudo-code implementation of the personalized hashing trick. Note that with the personalized hashing trick, using a hash function $h : \{Strings\} \rightarrow [1..b]$, we only require b parameters independent of how many users or words appear in our system.

personalized_hashingtrick(string *userid*, [string] *email*)
  $\vec{x} = \vec{0}$
  for *word* **in** *email* **do**
    $i = h(word)$
    $\vec{x}_i = \vec{x}_i + \xi(word)$
    $j = h(word\ userid)$
    $\vec{x}_j = \vec{x}_j + \xi(\text{word} \circ \text{userid})$
  end for
return $\vec{x}$

3. EXPERIMENTAL SET-UP AND RESULTS

To assess the validity of our proposed techniques, we conducted a series of experiments on the freely distributed

[2] We use the ∘ symbol to indicate string concatenation.

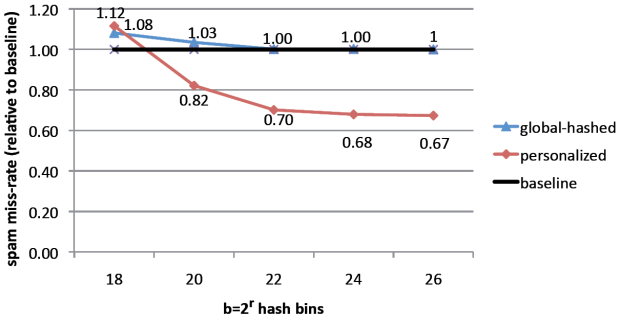


Figure 2: The results of the global and hybrid classifiers applied to a large-scale real-world data set of 3.2 million emails.

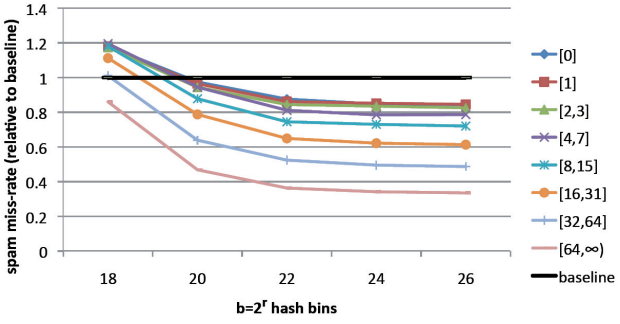


Figure 3: The amount of spam left in users' inboxes, relative to the baseline. The users are binned by the amount of training data they provide.

trec07p benchmark data set, and on a large-scale proprietary data set representing the realities of an open-membership email system. The trec data set contains 75,419 labelled and chronologically ordered emails taken from a single email server over four months in 2007 and compiled for trec spam filtering competitions [4]. Our proprietary data was collected over 14 days and contains $n = 3.2$ million anonymized emails from $|U| = 400{,}000$ anonymized users. Here the first ten days are used for training, and the last four days are used for experimental validation. Emails are either spam (positive) or ham (non-spam, negative). All spam filter experiments utilize the Vowpal Wabbit (VW) [5] linear classifier trained with stochastic gradient descent on a squared loss. Note that the hashing trick is independent of the classification scheme used; the hashing trick could apply equally well with many learning-based spam filtration solutions. To analyse the performance of our classification scheme we evaluate the spam catch rate (SCR, the percentage of spam emails detected) of our classifier at a fixed 1% ham misclassification rate (HMR, the percentage of good emails erroneously labelled as spam). We note that the proprietary nature of the latter data set precludes publishing of exact performance numbers. Instead we compare the performance to a baseline classifier, a global classifier hashed onto $b = 2^{26}$ dimensions. Since 2^{26} is far larger than the actual number of terms used, $d = 40$M, we believe this is representative of full-text classification without feature hashing.

4. THE VALIDITY OF HASHING IN EMAIL SPAM FILTERING

To measure the performance of the hashing trick and the influence of aggressive dimensionality reduction on classifier quality, we compare global classifier performance to that of our baseline classifier when hashing onto spaces of dimension $b = \{2^{18}, 2^{20}, 2^{22}, 2^{24}, 2^{26}\}$ on our proprietary data set. The results of this experiment are displayed as the blue line in Figure 2. Note that using 2^{18} bins results in only an 8% reduction in classifier performance, despite large numbers of hash collisions. Increasing b to 2^{20} improves the performance to within 3% of the baseline. Given that our data set has 40M unique tokens, this means that using a weight vector of 0.6% of the size of the full data results in approximately the same performance as a classifier using all dimensions.

Previously, we have proposed using hybrid global/personal spam filtering via feature hashing as a means for effectively mitigating the effects of differing opinions of spam and ham amongst a population of email users. We now seek to verify the efficacy of these techniques in a realistic setting. On our proprietary data set, we examine the techniques illustrated in Section 2 and display the results as the red line in Figure 2. Considering that our hybrid technique results from the cross product of $|U| = 400$K users and $d = 40$M tokens, a total of 16 trillion possible features, it is understandable that noise induced by collisions in the hash table adversely affects classifier performance when b is small. As the number of hash bins grows to 2^{22}, personalization already offers a 30% spam reduction over the baseline, despite aggressive hashing.

In any open email system, the number of emails labelled as either spam or non-spam varies greatly among users. Overall, the labelling distribution approximates a power law distribution. With this in mind, one possible explanation for the improved performance of the hybrid classifier in Figure 2 could be that we are heavily benefiting those few users with a rich set of personally labelled examples, while the masses of email users – those with few labelled examples – actually suffer. In fact, many users do not appear at all during training time and are only present in our test set. For these users, personalized features are mapped into hash buckets with weights set exclusively by other examples, resulting in some interference being added to the global spam prediction.

In Section 2, we hypothesized that using a hybrid spam classifier could mitigate the idiosyncrasies of the most active spam labellers, thereby creating a more general classifier for the remaining users, benefiting everyone. To validate this claim, we segregate users according to the number of training labels provided in our proprietary data. As before, a hybrid classifier is trained with $b = \{2^{18}, 2^{20}, 2^{22}, 2^{24}, 2^{26}\}$ bins. The results of this experiment are seen in Figure 3.

Note that for small b, it does indeed appear that the most active users benefit at the expense of those with few labelled examples. However, as b increases, therefore reducing the noise due to hash collisions, users with no or very few examples in the training set also benefit from the added personalization. This improvement can be explained if we recall the subjective nature of spam and ham – users do not always agree, especially in the case of business emails or newsletters. Additionally, spammers may have infiltrated the data set with malicious labels. The hybrid classifier absorbs these peculiarities with the personal component, freeing the global component to truly reflect a common definition of spam and ham and leading to better overall generalization, which benefits all users.

5. MITIGATING THE ACTIONS OF MALICIOUS USERS WITH HYBRID HASHING

In order to simulate the influence of deliberate noise in a controlled setting we performed additional experiments on the trec data set. We chose some percentage, *mal*, of 'malicious' users uniformly at random from the pool of email receivers, and set their email labels at random. Note that having malicious users label randomly is actually a harder case than having them label adversarially in a consistent fashion – as then the personalized spam filter could potentially learn and invert their preferences.

Figure 4 presents a comparison of global and hybrid spam filters under varying loads of malicious activity and different sized hash tables. Here we set $mal \in \{0\%, 20\%, 40\%\}$. Note that malicious activity does indeed harm the overall spam filter performance for a fixed classifier configuration. The random nature of our induced malicious activity leads to a 'background noise' occurring in many bins of our hash table, increasing the harmful nature of collisions. Both global and hybrid classifiers can mitigate this impact somewhat if the number of hash bins b is increased. In short, with malicious users, both global (dashed line) and hybrid (solid line) classifiers require more hash bins to achieve near-optimum performance. Since the hybrid classifier has more tokens, the number of hash collisions is also correspondingly larger. Given a large enough number of hash bins, the hybrid classifier clearly outperforms the single global classifier under the malicious

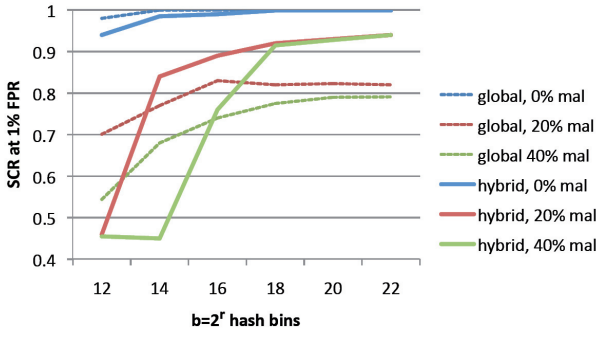


Figure 4: The influence of the number of hash bins on global and hybrid classifier performance with varying percentages of malicious users.

settings. We do not include the results of a pure local approach, as the performance is abysmal for many users due to a lack of training data.

6. CONCLUSION

This work demonstrates the hashing trick as an effective method for collaborative spam filtering. It allows spam filtering without the necessity of a memory-consuming dictionary and strictly bounds the overall memory required by the classifier. Further, the hashing trick allows the compression of many (thousands of) classifiers into a single, finite-sized weight vector. This allows us to run personalized and global classification together with very little additional computational overhead. We provide strong empirical evidence that the resulting classifier is more robust against noise and absorbs individual preferences that are common in the context of open-membership spam classification.

REFERENCES

[1] Caruana, R. Algorithms and applications for multitask learning. Proc. Intl. Conf. Machine Learning, pp.87–95. Morgan Kaufmann, 1996.

[2] Attenberg, J.; Weinberger, K.; Dasgupta, A.; Smola. A.; Zinkevich, M. Collaborative email-spam filtering with the hashing trick. Proceedings of the Sixth Conference on Email and Spam, CEAS 2009, 2009.

[3] Weinberger, K.; Dasgupta, A.; Attenberg, J.; Langford, J.; Smola, A. Feature hashing for large scale multitask learning. ICML, 2009.

[4] Cormack, G. TREC 2007 spam track overview. The Sixteenth Text REtrieval Conference (TREC 2007) Proceedings, 2007.

[5] Langford, J.; Li, L.; Strehl, A. Vowpal Wabbit online learning project. http://hunch.net/?p=309, 2007.

PRODUCT REVIEW

MICROSOFT SECURITY ESSENTIALS

John Hawes

Microsoft Security Essentials, the long-awaited replacement for the *Windows Live OneCare* package, is finally with us. When the globe-straddling giant began its push into the anti-malware sphere a few years back, some initial poor test results dented the launch of *OneCare*, and sluggish performance and general lack of user interest were immediately apparent. Despite a much improved version 2, *Microsoft* quickly decided to give up on the product – and its retirement and replacement were announced almost a year ago. The new project, codenamed 'Morro', excited a great deal of debate and no little controversy around the topic of free AV – and the debate continues to rumble on now that *Security Essentials* is available to the public.

One question which has received little attention amongst the hype and hyperbole surrounding the launch of *Security Essentials* is: what is meant by 'free AV'? We know from correspondence with our readers that some people always run some high-quality, multi-function suite without paying for it. They do so quite legally, simply by switching from one free trial to another every few months. Others get free access to security software via a value-added model. Should I need one, I could get myself set up with an expensive security suite at no charge thanks to extras thrown in with my bank account, my ISP, my phone line or TV provider, or any of a number of others.

Of course, none of these are completely free – the trial option requires considerable investment of effort, while the value-added path depends on having paid for the original product or service. What about the people who don't have the time or inclination to regularly reinstall software, and who can't afford the kind of service contracts that throw in extras? For them, the most common answer is the free-for-home-use model, under which developers make pared-down versions of their software available free of charge, on the condition that it is only for personal use.

The free-for-home-use field is currently dominated by the three 'A's: *Alwil* (*avast!*), *AVG* and *Avira*. Many other providers also release parts of their product range without charge, but it is these big three which dominate. According to a recent blog post by *Alwil*'s CEO, 50% of the world's 500 million consumer machines run one of the three, compared to 20% running one of the 'market-leading' brands, *Symantec*, *McAfee*, *Trend Micro* and *Kaspersky*[1]. The business model is a simple one, providing the companies with wide market penetration and excellent brand recognition. A small fraction of users of free products will upgrade to paid editions, while the widespread distribution of the free products also provides the companies with an additional source of fresh samples that's pretty hard to beat. All this can be achieved for no more than the cost of a little server space and bandwidth for updates, and a few man-hours for the monitoring of help forums (which are essentially fan-maintained).

So where will *Security Essentials* find its place? Clearly *Microsoft* has slightly different goals here – the firm has little need of additional publicity, and it seems unlikely that many users of *Security Essentials* will be tempted to upgrade to its corporate big brother *Forefront* (*Security Essentials* is targeted squarely at home users). There will, of course, be some advantage to be gained from the increase in new samples flowing into the company's labs, but with such massive presence in all sorts of areas, this should not make a great impact.

The stated aim of the product is to provide protection for those users currently running their machines unprotected, particularly in less developed nations (some reports have estimated as many as 50% of users are not running up-to-date security software[2]). As always when *Microsoft* takes steps in the security world however, many sceptical commentators have viewed such altruistic claims with suspicion, muttering that such efforts may simply be part of an ongoing campaign to counterbalance the company's reputation for insecurity in its operating systems. Despite making great strides in recent releases, that reputation lingers thanks to the continual discovery of new flaws and vulnerabilities in many areas – perhaps inevitably, given the breadth and scope of the company's product range, and the armies of hackers beavering away looking for cracks to crowbar open.

Once the decision to retire *OneCare* had been taken, releasing the *Security Essentials* product seemed almost to have no downside for *Microsoft*. The interface will likely require little maintenance, support will likely be kept to a minimum, while the engine teams and malware analysts will already be hard at work maintaining *Forefront* – so it could just be that the proclaimed altruism is genuine, and the improvement of the firm's security image just a side effect. There are similarly few downsides for users – the product is available free of charge to those who want it, and even if it is only taken up by a tiny fraction of potential users and only provides minimal protection, it will still contribute positively to the overall security picture. The uptake may well depend on the quality of the product, but will also doubtless be influenced by promotion, marketing and the response of users.

[1] http://blog.avast.com/2009/10/02/and-what-about-microsoft-security-essentials%e2%80%94mse/

[2] http://www.theregister.co.uk/2009/09/29/ms_security_essentials/

SECURITY ESSENTIALS: PROMOTION, INFORMATION AND SUPPORT

The initial announcement of the retirement of *OneCare* and its replacement with a free offering attracted considerable interest, and a public beta release was heavily oversubscribed with many more users trying to get hold of it than expected. However, the final full release came with less of a fanfare, with interest somewhat depleted after almost a year of waiting, a staggered release into different territories and the impending release of *Windows 7* all factors in the relative quietness of the product's emergence.

Considerable attention was paid by the technical branches of the media however, with a mixed bag of early reviews appearing in the weeks following the official public launch. Many commented favourably on the product's simplicity and ease of use, while some – rather unfairly – criticized the absence of the full range of additional layers of protection, such as firewall and HIPS protection, found at the more advanced end of the suite market. Despite the somewhat tepid reaction, downloads were reported to have reached 1.5 million within the first week.

The product can be acquired from the microsite located at www.microsoft.com/Security_Essentials. The landing page is pretty simple and straightforward, with a big download button taking centre stage along with some basic information about the product. The fact that the product is free only for home users of genuine licensed copies of *Windows* is clearly highlighted; business users are directed to *Forefront*, while those seeking more information on malware are provided with a link to the Malware Protection Center (MMPC) portal. A selection of certification badges appeared soon after the site went live.

The most prominent buttons on the page are for help, support and an installation video (the use of which involved allowing scripting in the browser and, to view the video, installation of *Microsoft*'s *Silverlight* system). An option was provided to download the video as a WMV file and watch it the old-fashioned way, but it appeared to be beyond the abilities of several older copies of *Windows Media Player*[3].

A resources tab provides more links to the MMPC, the EULA and privacy policy (for the SpyNet system, which we'll come to later), and the system requirements – the product claims support for *Windows* versions from *XP SP2* up via *Vista* to the new *Windows 7*, and needs 140MB of hard drive space, a 500MHZ processor and 512MB of RAM (rising to 1GHz and 1GB for the *Vista* and *Windows 7* version). This all seems pretty reasonable, given the requirements of the operating systems, and hopefully few users will find themselves unable to use the product through a lack of computing power.

The final part of the page – probably the most important once users have acquired and installed the product – is the support area. This is already well stocked with FAQs, guides and how-tos alongside the videos mentioned earlier, and also provides access to a community-driven forum for resolving more specific problems. This seems to have generated some useful content already, and appears to be well staffed by some helpful and knowledgeable pros alongside the user community. With no proper manual apparent, there seems to be ample information available to steer users through any potential issues they might have, while a proper online support-case submission system is also provided for more troublesome matters, alongside a new threat reporting system.

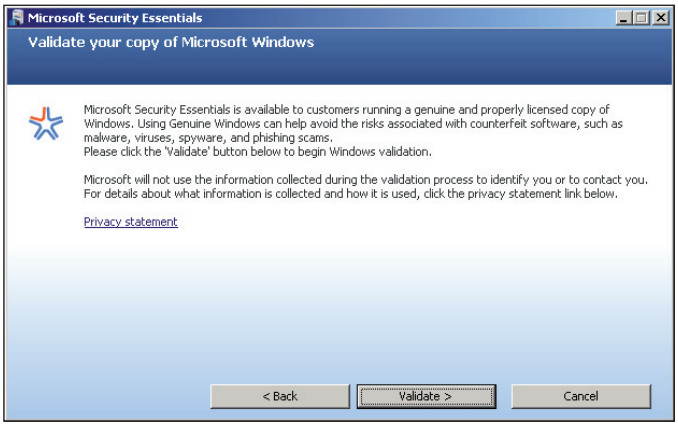
The whole thing is smooth and slick (some issues with the videos notwithstanding), but there was one thing that did stand out like a sore thumb throughout the site: just about every page appears to carry advertising. Some layer of ad filtering managed to block the advertising on most systems we used to visit the site, but this spoiled the smooth clean lines with clunky block messages. I suppose we could have allowed the ads to see whether they would fit in nicely with the look and feel of the surroundings. Either way, though, the idea that help – even for a free product – is a suitable place for advertising seems rather uncomfortable, and somewhat dents the supposed altruistic ethos behind the project.

USER EXPERIENCE

With the product downloaded – a small initial file which took seconds to download with a fast connection – the installation process is pretty straightforward. It sails lightly and quickly through the standard set-up steps, with the only less usual one being a check for the genuineness of *Windows*. As honest, well-behaved people we did not have any unlicensed or pirate copies of *Windows* to hand to observe how it would respond in such circumstances, but on legitimate systems it trips through nice and quickly (even more so if the Genuine Advantage set-up has already been performed). The check for the legitimacy of the running *Windows* system has proved controversial, with some commentators arguing that the bulk of the audience *Microsoft* is aiming for – those running unprotected systems in less advanced regions – are likely also to be running unlicensed copies of *Windows*. However, the decision not to allow these users to protect themselves does make some business sense.

At the end of the installation process the product connects back to base to update itself, and offers to run a full system

[3] Further product information and videos are at http://www.microsoft.com/presspass/presskits/microsoftsecurityessentials/materials.aspx



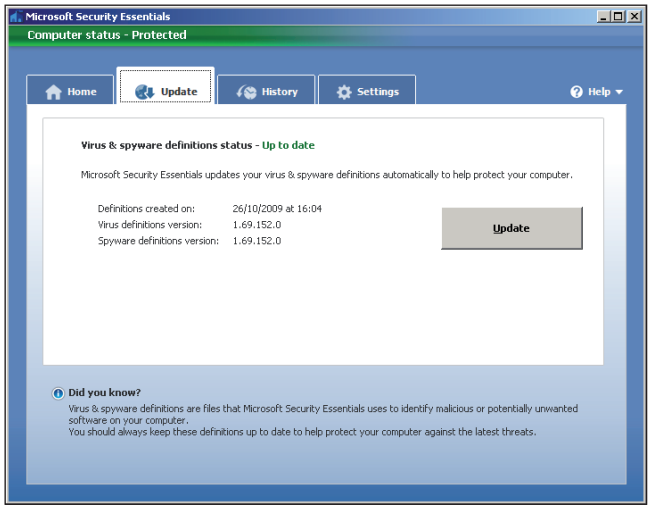
scan once it is complete. In our tests the update never took more than a couple of minutes, even with a less than ideal web connection or having waited several weeks after the initial download. Once installed, a rather lumpy icon (which, after a few moments of staring, we deduced represented a square castle flying a large flag) appears in the system tray to indicate protection is in place.

The product itself is remarkably simple and clear, with a main page offering bare data on the protected status of the system (a bar turning red if any kind of threat has been observed), and a few buttons for different kinds of on-demand scans. An update tab provides some information on update status and a button to run a fresh update; a history tab reports details of threats detected and how they have been treated; and a settings tab provides some configuration options – with rather more choice available than might have been expected. There is a sensible set of defaults and a selection of useful options – including the exclusion of certain areas, types of files and even running processes from scanning – but there is not quite the in-depth configuration available as seen in the most sophisticated products. The scheduler allows only a single job, and as in so many systems is set to run in the middle of the night on a Sunday; some users may want to adjust this, particularly if using laptops or saving energy by shutting down at such times.

The most interesting part of the settings tab are the controls for the 'SpyNet' system, yet another online community-reporting system designed to gather information on what is being detected. The default 'basic' setting provides *Microsoft* with minimal details of any detection that occurs, mainly limited to what was detected and when. Meanwhile the 'advanced membership' option allows the product to upload much more detailed information including filenames and locations of dangerous files. There is no option to run without reporting data to base, which may worry some users from a privacy angle, but in most circumstances even the advanced setting is unlikely to breach anyone's privacy. As such systems help measure and monitor the malware problem, providing useful information on how infections function and spread, I would recommend that any user not crippled by paranoia enable the advanced mode.

Running the product for a while and putting it through our standard set of speed measurements showed it had a reasonably low footprint, not interfering with normal usage even while scanning, thanks to judicious use of prioritization. A full set of speed measurements were taken in direct comparison with the last VB100 test – which was admittedly on an unsupported server platform, but this seemed not to impede the product's operation and showed that it fitted fairly neatly into the upper end of the range for speed and overheads, with both on-demand and on-access speeds around a third faster than those recorded for big brother *Forefront* across the board. We got similar results on *XP* and *Vista*, and should have more accurate and complete results available, including details of *Windows 7* performance, once the product has been through a full VB100 comparative in a month's time. As well as standard on-access and on-demand scanning, the product also checks web downloads and email attachments specifically as they arrive by integration with standard *Windows* functions for this. Here, a noticeable but totally acceptable additional delay was added to the download.

We did note a fairly hefty slowdown of the system in a couple of cases, particularly when running on low-end netbooks at the bottom end of the supported power range. This was most evident during boot-up and recovering from hibernation, and was quickly diagnosed – when the product is installed, its updating system is integrated with *Windows Update*, which on some systems I have found it best to run manually thanks to this slowdown during boot-up as new patches are downloaded. Enabling the update system for the malware definitions also enables the full *Windows* updates, and led to the problems noted. In most cases of course, users should always have *Windows Update* active to ensure



they get the latest security fixes as quickly as possible, so this will not affect most users.

Having surveyed the product in its normal, fairly dormant state, it was time to see how well it fared in more challenging times.

PROTECTION CAPABILITIES

Continuing the process from the speed testing, we pushed the product through the full set of VB100 tests from the previous comparative (see *VB*, October 2009, p.17), albeit with definitions around a month newer than the official test deadline. We found detection scores as expected closely mirroring those achieved by *Forefront*, with the newer samples in the RAP sets covered much better thanks to the intervening time since the sets were gathered. Across the full trojan set and all of the RAP sets detection rates were steady and reliable, never dipping below 95%. The other standard sets were handled pretty impeccably, with no issues in the WildList set despite even larger numbers of highly complex polymorphic strains added in recent weeks. Several other variants of W32/Virut, which continues to show up high on our prevalence charts, were also tested and detected perfectly. Stability was excellent and the product behaved impeccably throughout, even when handling sets of strange and malformed files known to cause problems for some engines. We also liked the way the on-access scanner does not feel the need to bombard the user with alert pop-ups, instead going about its business simply and quietly and recording malicious activity in its main interface, to be reviewed at leisure.

There was one minor oddity in the history set-up though – one that perhaps is unique to the likes of us. Having run a scan of our full sets, we opted not to let the product plough slowly through the whole lot removing, cleaning and quarantining tens of thousands of files, so simply closed the scan window. On checking the history area later, although it claimed to contain details of all threats detected, it reported nothing. We later found that reports seemed to make their way into the history only if they have been acted on in some way – either cleaned or allowed. As this is the default and probably most sensible way to operate, it seems unlikely that this will affect most users, but it is still perhaps a little misleading.

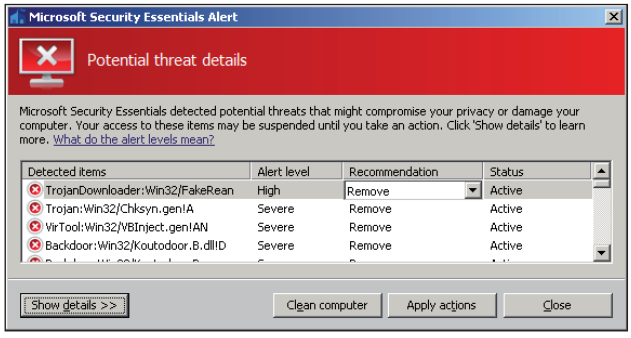
As well as the test sets used in the last comparative, we also built an updated RAP-style set from samples received shortly before and just after installing the product. Once again we saw some excellent detection rates – particularly impressive in the week +1 set which contained mostly items not seen by the labs before. The strong team being built up by *Microsoft* for its anti-malware department has already had great success as the scores achieved by the company's products, both in our own and other independent tests, have steadily increased in the last year or so to reach some extremely competitive levels – this trend shows no sign of abating any time soon. Our next test was to run the product against samples from the latest WildList, which again caused no problems.

Scanning our full clean sets produced no false positives, even in the batches of samples deemed too obscure or bizarre to include in our official set. *Microsoft*'s products have shown a pretty clean performance in this area in our tests for some time now, with not a single false positive since *OneCare* first entered the VB100 several years ago – a pretty strong achievement perhaps helped by the firm's massive software certification programmes and penetrating ability to see what people are running on systems around the world.

All in all the basics of straightforward static anti-malware protection seemed to be provided to a very satisfactory standard. The set of tests we performed would easily have qualified the product for a VB100 award had it been under official conditions, and it looks pretty likely that, barring surprise disasters, certification will be achieved at the first attempt (which will hopefully be in the upcoming *Windows 7* test).

Cleaning seemed pretty solid too, with most of the handful of items we tested removed without difficulty. As is usually the case, a few innocuous remnants were left behind in some cases, along with a few registry entries and changes to the hosts file – this is par for the course though and few products will be brave enough to remove all possible side effects of some attacks for fear of damaging important existing settings.

Moving on to other features, we learned early on from the product literature and early reviews that there is a little more than basic protection here, with the SpyNet reporting system also featuring a level of cloud-based intelligence and even some behavioural monitoring, apparently watching unknown files for suspect behaviour, checking



for confirmations from other sources and automatically blocking new threats. We endeavoured to test this behaviour, having found a selection of newly gathered samples that were not being detected by the signature scanner which we ran on sacrificial systems with limited networking to allow the product to connect out while minimizing the potential danger of the malware. Unfortunately, despite our best efforts, we could find no confirmation of any additional protection provided by the cloud-based system – with each of the samples not spotted by the straight scanner allowed to operate as they pleased. We will keep an eye on this new technology and see how it develops.

CONCLUSIONS

Overall, *Security Essentials* proved a pretty decent product for the price bracket. Some commentators have complained about the lack of additional suite-type features, but there are still many similar standard anti-malware products on the market, with great demand for such protection even if paid for. The ability to pick and choose protective components remains important to many, while the prices of the more complete suites put them out of range for many others. We tried the product in combination with a number of third-party firewalls, spam filters and even behavioural monitors, with no sign of any clash or incompatibility, and we were able to provide ourselves with a fairly decent sense of security without spending a penny. Of course this takes some effort and perhaps a little skill, and for many users *Security Essentials* will in fact be their security be-all and end-all; while such an approach is not to be encouraged, it is far preferable to having no security at all.

Providing an extremely respectable level of detection of known malware, some top-notch heuristic and generic coverage of emerging threats, and a simple approach to usability which shouldn't baffle even the most technophobic, *Security Essentials* makes a strong addition to the line-up of free solutions on the market. If it is indeed taken up by the vast armies of unprotected systems out there, it should make quite some dent in the number of zombie systems attacking and spamming the rest of us.

COMPARATIVE REVIEW

ANTI-SPAM COMPARATIVE REVIEW

Martijn Grooten

This month's anti-spam comparative review saw yet another increase in the field of competitors with 14 products taking their place on the test bench; the same 12 products that participated in the September test were joined by two new ones. One of the new products is the first anti-spam solution to take part in our test that runs on a virtual machine – demonstrating yet another possibility for administrators searching for a decent anti-spam solution to run in their organization. The 12 VBSpam awards given out this month – another record – demonstrate that there is plenty of choice when it comes to very good solutions.

THE TEST SET-UP

No changes were made to the test set-up, apart from some modifications to the corpora used, as is explained below. As usual, the full methodology can be found at http://www.virusbtn.com/vbspam/methodology/.

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. Those running on *Linux* ran on *SuSE Linux Enterprise Server 11*; the *Windows Server* products ran either the *2003* or the *2008* version, depending on which was recommended by the vendor.

THE EMAIL CORPUS

The test ran from 1pm UK time on 16 October 2009 to 12pm UK time on 30 October 2009 – with the end of British Summer Time coming in the middle of the test, this meant the test ran for two weeks exactly. The corpus contained a total of 199,842 emails: 2,121 ham messages and 197,721 spam messages. The latter consisted of 176,667 messages provided by Project Honey Pot and 21,054 spam messages sent to @virusbtn.com addresses.

The ham emails consisted of all legitimate emails sent to @virusbtn.com addresses. This time, however, some senders were excluded from the test set: these were the senders of emails that regularly discuss spam- and malware-related topics (for example anti-spam discussion lists) and as such *regularly* contain links to malicious and/or spamvertised URLs. We believe that not only are such emails unlikely to occur in the legitimate email stream of an average organization, but also that the recipients of such emails generally have the level of knowledge and technical ability required to whitelist these particular senders. All

emails from these senders were removed from the test set, regardless of the contents of the individual emails. (Of course, it is possible that other legitimate senders also included malicious and/or spamvertised URLs in their emails – however, these were not excluded from the test set.)

Unsurprisingly, this affected the products' false positive rates and only one product blocked more than one per cent of all legitimate emails in the test. Interestingly, no legitimate email was blocked by more than four products – so while developers might argue that certain emails are hard to recognize as legitimate, it can also be pointed out that for every email they incorrectly blocked, there were at least ten other products that correctly recognized it as ham.

To make up for the exclusion of some senders, we subscribed some of our addresses to a number of email discussion lists. We believe this has several advantages: firstly, it adds to the variety of topics discussed in the ham stream, as well as to the variety of sending domains and IP addresses, and thus makes the test results more representative for an average company. Secondly, these emails are generally very much wanted by their recipients and as such do not fall in the grey area of legitimate-yet-not-particularly-wanted emails. And thirdly, because we can (and will) vary the lists subscribed to over time, we can give the full contents of the emails to developers whose products blocked them – in doing so neither compromising our own confidentiality nor introducing the possibility for developers to whitelist these senders and thus gain unfair advantage over their competitors. Finally, it should be noted that spam is occasionally sent to discussion lists – for instance when a subscriber's email account has been compromised. This happened once during the running of the test and this email was classified as spam.

RESULTS

In previous reviews we have published both the overall false positive (FP) rate and the false positive rate as a ratio of the total *VB* mail stream – the latter number is of little practical use, but has been included in the past for reference. However, because of the modifications described above, the mail corpora used are not those of a real company and therefore we have decided to leave this FP ratio out of the report; interested readers will still be able to compute the ratio themselves.

BitDefender Security for Mail Servers 3.0.2

SC rate (total): 97.89%
SC rate (Project Honey Pot corpus): 98.90%
SC rate (VB spam corpus): 89.37%
FP rate: 0.707%

Two interesting papers presented at VB2009 demonstrated that *BitDefender* does more than simply use existing technologies to fight spam: the developers in the company's Bucharest-based anti-spam lab are working hard to find new ways to stay ahead of the spammers. The product has won a VBSpam award in each of the three previous anti-spam tests and while this month the spam catch rate is slightly lower than that of the previous test, it is still sufficient for the product – again, the *Linux* version – to win a VBSpam Gold award.

(Note: In the previous test report it was stated that *BitDefender* had 11 false positives. Careful investigation of these showed that a mistake was made and one reported false positive should not have been counted as such. This did not affect the level of the award earned by the product.)

Fortinet FortiMail

SC rate (total): 98.47%
SC rate (Project Honey Pot corpus): 98.98%
SC rate (VB spam corpus): 94.12%
FP rate: 0.047%

FortiMail, a hardware appliance from Canadian company *Fortinet*, won a VBSpam Silver award in the two previous tests and while not entirely unhappy with that, its developers believed the product was capable of doing better. For this test, the product's spam criteria were loosened in an attempt to reduce the false positive rate (which, so far, has prevented it from winning a higher level award), while an upgrade of the firmware was intended to help maintain a high spam catch rate. The latter worked very well, but even more impressive was the product's low false positive rate: out of well over 2,000 emails, only one newsletter was missed. A VBSpam Platinum award is well deserved and the developers' faith in their product fully justified.

Kaspersky Anti-Spam 3.0

SC rate (total): 97.52%
SC rate (Project Honey Pot corpus): 98.58%
SC rate (VB spam corpus): 88.65%
FP rate: 0.141%

In previous reports I have lauded *Kaspersky*'s anti-spam solution for the minimal maintenance it requires: it is installed on a *Linux* machine and works straight away. Of

course 'works' doesn't necessarily mean 'works well', but it does in the case of *Kaspersky*. Particularly impressive is the product's consistently low false positive rate – only three emails were incorrectly blocked during the test. This combined with a good spam catch rate earns the product yet another VBSpam Gold award.

McAfee Email Gateway (formerly IronMail)

SC rate (total): 99.02%
SC rate (Project Honey Pot corpus): 99.85%
SC rate (VB spam corpus): 92.00%
FP rate: 0.707%

Like last time, *McAfee*'s *Email Gateway* appliance (also sold under its former name *IronMail*) was the only product that scanned and, in cases of suspected spam, blocked emails during the SMTP transaction, with only the harder-to-filter emails being scanned at a later stage. This solution worked well: the product once again had a very high spam catch rate. The false positive rate was significantly lower than on the last occasion and all but a few of these false positives were scanned at a later stage; in a real scenario these emails would probably have been stored in quarantine rather than being discarded altogether. With still a few too many false positives for a platinum award, the product won its second consecutive VBSpam Gold award.

McAfee Email and Web Security Appliance

SC rate (total): 98.75%
SC rate (Project Honey Pot corpus): 99.28%
SC rate (VB spam corpus): 94.36%
FP rate: 0.189%

'Never change a winning formula', they must have thought at *McAfee* and in a system administrator's ideal scenario the appliance – the only product to win a VBSpam Platinum award in the last test – was run using exactly the same set-up. This scenario worked well for the product and combining a very low false positive rate with a very high spam catch rate, it won its second consecutive VBSpam Platinum award.

M86 MailMarshal SMTP

SC rate (total): 99.62%
SC rate (Project Honey Pot corpus): 99.94%
SC rate (VB spam corpus): 96.92%
FP rate: 0.519%

The brand *M86 Security* has been around in the world of computer security for barely two months; before that the company was known as *Marshal8e6*, which in turn was the merger of *Marshal* and *8e6*. The company offers a number of security solutions including its *MailMarshal SMTP* spam filter.

This product, which comes with its own MTA and was run on *Windows Server 2003*, uses a multi-layered approach where an email has to pass several tests before it is sent to the user's inbox. Among these tests are SpamBotCensor, which uses knowledge about the engines used by various spam bots to detect spammers at the SMTP level, and SpamCensor, which uses heuristics to block spam based on the contents of the email. The product's user interface gives the administrator plenty of opportunities to modify the rules for the various tests and can easily be fine-tuned to meet the needs of a particular organization.

Unfortunately, the SpamBotCensor could not be applied during our test, but *MailMarshal* still had the highest spam catch rate of all participating products. Combined with a low false positive rate, it just missed out on a platinum-level award; a VBSpam Gold award nevertheless marks an excellent debut for *MailMarshal*.

MessageStream

SC rate (total): 99.49%
SC rate (Project Honey Pot corpus): 99.82%
SC rate (VB spam corpus): 96.64%
FP rate: 0.471%

One reason why organizations may want to choose a hosted anti-spam solution is the little maintenance it requires. That is certainly the case with *MessageStream*, the hosted solution provided by *Giacom*. Without a lot of intervention from the developers it achieved yet another very high spam catch rate and missed out on a platinum award by just a few emails; it is the only product to have won four VBSpam Gold awards in a row.

Messaging Architects M+Guardian

SC rate (total): 98.75%
SC rate (Project Honey Pot corpus): 99.26%
SC rate (VB spam corpus): 94.47%
FP rate: 0.943%

It is always disappointing to see a product win a lower-level award in a test than in the previous one. In reality, the *M+Guardian* appliance performed better on this occasion than in the last test – however, since the thresholds have become stricter the product's fourth VBSpam award is a silver one. It will be interesting to see whether the product will be able to do better again next time around.

Microsoft Forefront Protection 2010 for Exchange Server

SC rate (total): 99.00%
SC rate (Project Honey Pot corpus): 99.46%
SC rate (VB spam corpus): 95.16%
FP rate: 0.471%

Few will have been awaiting this review more eagerly than the developers at *Microsoft*: their *Forefront* product won a VBSpam Silver award in its first test in September. At the time the product was still a release candidate, and in the weeks following that test they believed some issues had been solved – thus they were eager to see if the changes had made an improvement. They had: the product's false positive rate was reduced by almost four-fifths compared to the last test, while it maintained a high spam catch rate. A VBSpam Gold award will be an extra reason to celebrate the official release of the product in the second week of November.

Sanesecurity signatures for ClamAV

SC rate (total): 72.40%
SC rate (Project Honey Pot corpus): 73.24%
SC rate (VB spam corpus): 65.34%
FP rate: 0.33%

In previous reviews it has not been made clear enough that while the *Sanesecurity* signatures work together with *ClamAV*, they have little to do with that product (which is mainly an anti-malware product). Perhaps unsurprisingly for something that scans emails purely based on content, this product sees a greater fluctuation from day to day than other products; in this case it means that some 'bad days' in the first week of the test caused the product's final spam catch rate to be significantly lower than during the previous test. Still, for what is only a partial solution – which would be an effective part of a multi-layered solution – a spam catch rate of well over 70% is a rather good score, although a number of false positives caused by incorrectly blacklisted URLs demonstrate that the product isn't entirely without fault either.

SPAMfighter Mail Gateway

SC rate (total): 97.22%
SC rate (Project Honey Pot corpus): 97.36%
SC rate (VB spam corpus): 96.10%
FP rate: 0.66%

SPAMfighter's *Mail Gateway* debuted in the previous VBSpam test, but failed to win an award. The developers at the Danish company believed this may have been the result of the product being set up in a manner that was less than ideal for our test; they also believed their product might have been disproportionately disadvantaged by issues with the network. While these issues were solved, the product was set to filter less stringently to reduce the number of false positives, while at the same time the linger filter was turned on. This filter will hold on to emails that aren't immediately recognized as either ham or spam and rescan them after a certain amount of time, by which time the content might be recognized by the updated spam filter. Of course, this may cause delays for legitimate email, but the filter can be set to work only at certain times of day (such as outside office hours), when delays aren't generally noted; in this test it was turned on 24 hours a day.

The changes certainly had a very positive effect on the product's performance: the false positive rate was reduced greatly and the spam catch rate was still rather good; the product performed almost equally well on both spam corpora, showing that its performance wasn't just luck. A VBSpam Gold award is well deserved.

SpamTitan

SC rate (total): 99.48%
SC rate (Project Honey Pot corpus): 99.97%
SC rate (VB spam corpus): 95.41%
FP rate: 0.377%

Spam filters are essential for any organization, but for smaller companies buying separate hardware for spam filtering might not always be an option. Running the filter on a virtual machine could then be a solution and *SpamTitan*, a company based on the Irish west coast, offers such a solution. The product can easily be installed under *VMware* – for larger organizations, the same product is available as an ISO image that contains a complete operating system – and works almost immediately after installing. That is not to say the spam rules cannot be customized to suit a particular organization's needs: a web interface lets the user customize many rules of the blended approach the product uses to fight spam. I was particularly charmed by the simple, yet accurate explanations of the various anti-spam rules.

The fact that this approach worked well to block spam can be seen from the spam catch rate – which was among the highest in this test. At the same time, the product had a very low false positive rate, missing out on a platinum award by just a single email; a VBSpam Gold award is more than deserved.

Vircom modusGate

SC rate (total): 94.01%

SC rate (Project Honey Pot corpus): 94.37%

SC rate (VB spam corpus): 90.92%

FP rate: 3.772%

Vircom's *modusGate* product has failed to win an award in the last two VBSpam tests, but its developers are working hard to fix the issues that they believe are the cause of the poor performance in our tests. Still, with a false positive rate of more than three per cent and a spam catch rate significantly lower than that of most of its competitors, we cannot but deny *Vircom*'s *modusGate* a VBSpam award on this occasion.

Webroot E-Mail Security SaaS

SC rate (total): 99.31%

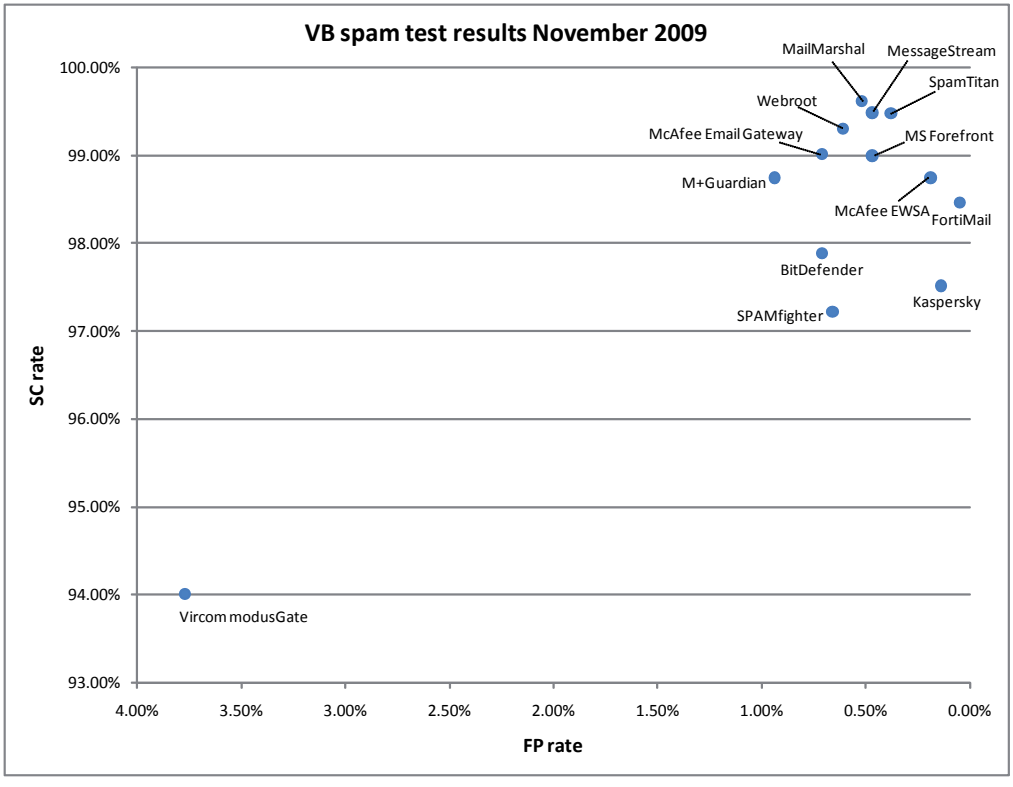
SC rate (Project Honey Pot corpus): 99.67%

SC rate (VB spam corpus): 96.31%

FP rate: 0.613%

Webroot only just missed out on a VBSpam Gold award in the last round of testing, winning its second VBSpam Silver award instead. Making small improvements is not a trivial task though, especially if competitors do the same thing and the thresholds thus become stricter. However, the developers of this hosted solution managed to improve their product enough to see the number of false positives reduced, while still having among the highest spam catch rates and thus this time *Webroot* earns a VBSpam Gold award.

				Total spam			Project Honey Pot spam			VB corpus		
	True negative	FP	FP rate	False negative	True positive	SC rate	False negative	True positive	SC rate	False negative	True positive	SC rate
BitDefender	2,106	15	0.71%	4,172	193,549	97.89%	1,935	174,732	98.90%	2,237	18,817	89.37%
FortiMail	2,120	1	0.05%	3,033	194,688	98.47%	1,794	174,873	98.98%	1,239	19,815	94.12%
Kaspersky	2,118	3	0.14%	4,904	192,817	97.52%	2,515	174,152	98.58%	2,389	18,665	88.65%
McAfee Email Gateway	2,106	15	0.71%	1,941	195,780	99.02%	257	176,410	99.85%	1,684	19,370	92.00%
McAfee EWSA	2,117	4	0.19%	2,466	195,255	98.75%	1,278	175,389	99.28%	1,188	19,866	94.36%
MailMarshal	2,110	11	0.52%	752	196,969	99.62%	103	176,564	99.94%	649	20,405	96.92%
MessageStream	2,111	10	0.47%	1,017	196,704	99.49%	310	176,357	99.82%	707	20,347	96.64%
M+Guardian	2,101	20	0.94%	2,472	195,249	98.75%	1,307	175,360	99.26%	1,165	19,889	94.47%
MS Forefront	2,111	10	0.47%	1,975	195,746	99.00%	955	175,712	99.46%	1,020	20,034	95.16%
Sanesecurity	2,114	7	0.33%	54,567	143,154	72.40%	47,269	129,398	73.24%	7,298	13,756	65.34%
SPAMfighter	2,107	14	0.66%	5,488	192,233	97.22%	4,667	172,000	97.36%	821	20,233	96.10%
SpamTitan	2,113	8	0.38%	1,025	196,696	99.48%	59	176,608	99.97%	966	20,088	95.41%
Vircom modusGate	2,041	80	3.77%	11,851	185,870	94.01%	9,940	166,727	94.37%	1,911	19,143	90.92%
Webroot	2,108	13	0.61%	1,358	196,363	99.31%	581	176,086	99.67%	777	20,277	96.31%



AWARDS

As in the previous test, the levels 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, the scores for any product with a false positive rate of more than 10% and/or a spam catch rate of less than 70% are removed from the computation of the averages; this did not apply to any of the products this month.

This month's benchmarks are then as follows:

- Platinum: SC 98.25%; FP 0.36%
- Gold: SC 95.60%; FP 0.71%
- Silver: SC 94.75%; FP 1.07%

The table shows the scores for all of the products on test. The highlighted columns show the scores used for the benchmark calculations. In the graph, *SaneSecurity* has been left out: this is only a partial solution and, as such, should not be compared directly with the other products.

CONCLUSION

The period between tests is used by developers to make improvements to their products. At the same time, we use this period to make improvements to the test set-up and to review our methodology. With the catch rates and (especially) the false positive rates of the various products edging closer to each other than ever, we believe that the way in which the product certifications are determined could do with some improvements. These changes will be announced in due course (well before the start of the next test) at http://www.virusbtn.com/vbspam.

The next test is set to run throughout December and the deadline for product submission will be 27 November 2009; any developers interested in submitting a product should email martijn.grooten@virusbtn.com. A number of new products have already committed to their participation and we are looking forward to an even bigger test.

December has traditionally been the month when spam levels rise to unprecedented heights, so it will be interesting to see which products are best at keeping their users' inboxes clean during the holiday period.

END NOTES & NEWS

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

A step-by-step masterclass in digital forensics and cybercrime will be run by ICFE on 19 November 2009 in Kuala Lumpur, Malaysia. The masterclass follows the launch of CSI Malaysia. See http://www.icfe-cg.com/.

ACSAC 2009 will be held 7–11 December 2009 in Honolulu, Hawaii. For details see http://www.acsac.org/.

The 26th Chaos Communication Congress (26C3) takes place 27–30 December 2009 in Berlin, Germany. The Congress offers lectures and workshops on a multitude of topics and attracts a diverse audience of hackers, scientists, artists and utopians from around the world. For more information see http://events.ccc.de/congress/2009/.

Black Hat DC 2010 takes place 31 January to 3 February 2010 in Washington, DC, USA. Online registration is now open. For details see http://www.blackhat.com/.

RSA Conference 2010 will be held 1–5 March 2010 in San Francisco, CA, USA. Registration is now open, with early bird discounted rates until 5 December 2009. For details see http://www.rsaconference.com/.

The 11th annual CanSecWest conference will be held 22–26 March 2010 in Vancouver, Canada. A call for papers is now open, with a submission deadline of 30 November 2009. For more details see http://cansecwest.com/.

The MIT Spam Conference 2010 is scheduled to take place 25–26 March 2010. A call for papers, venue announcements, and other details will be announced in due course at http://projects.csail.mit.edu/spamconf/.

Black Hat Europe 2010 takes place 12–15 April 2010 in Barcelona, Spain. A call for papers will open in January. See http://www.blackhat.com/.

Infosecurity Europe 2010 will take place 27–29 April 2010 in London, UK. For more details see http://www.infosec.co.uk/.

The 19th EICAR conference will be held 10–11 May 2010 in Paris, France with the theme 'ICT security: quo vadis?'. A call for papers has been issued, with submission deadlines of 20 December 2009 for peer-reviewed papers and 13 December for non-reviewed papers. For more information see http://www.eicar.org/conference/.

NISC11 will be held 19–21 May 2010 in St Andrews, Scotland. Interest in attending can be registered at http://nisc.org.uk/.

The 22nd Annual FIRST Conference on Computer Security Incident Handling takes place 13–18 June 2010 in Miami, FL, USA. The conference promotes worldwide coordination and cooperation among Computer Security Incident Response Teams and provides a forum for sharing goals, ideas and information on how to improve global computer security. For more details see http://conference.first.org/.

CEAS 2010 – the 7th annual Collaboration, Electronic messaging, Anti-Abuse and Spam Conference – will be held 13–14 July 2010 in Redmond, WA, USA. For details see http://ceas.cc/.

Black Hat USA 2010 takes place 24–29 July 2010 in Las Vegas, NV, USA. DEFCON 18 follows the Black Hat event, taking place 29 July to 1 August, also in Las Vegas. For more information see http://www.blackhat.com/ and http://www.defcon.org/.

The 19th USENIX Security Symposium will take place 11–13 August 2010 in Washington, DC, USA. For more details see http://usenix.org/.

VB2010 will take place 29 September to 1 October 2010 in Vancouver, Canada. For details of sponsorship opportunities and any other queries relating to VB2010, please contact conference@virusbtn.com.

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/